UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

DTIC FILE COPY

2

AD-A205 656

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE *(and Subtitle)* Ada Compiler Validation Summary Report: SoftTech Inc., Ada 86 Version 3.21, VAX 11/780-11/785 (Host) to Intel iAPX 80386P (Target)		5. TYPE OF REPORT & PERIOD COVERED 8 July 1988 to 8 July 1988
		6. PERFORMING ORG. REPORT NUMBER -
7. AUTHOR(s) National Bureau of Standards Gaithersburg, MD		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION AND ADDRESS National Bureau of Standards Gaithersburg, MD		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Ada Joint Program Office United States Department of Defense Washington, DC 20301-3081		12. REPORT DATE
		13. NUMBER OF PAGES
14. MONITORING AGENCY NAME & ADDRESS *(If different from Controlling Office)* National Bureau of Standards Gaithersburg, MD		15. SECURITY CLASS *(of this report)* UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A

16. DISTRIBUTION STATEMENT *(of this Report)*

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20 if different from Report)*

UNCLASSIFIED

18. SUPPLEMENTARY NOTES

19. KEYWORDS *(Continue on reverse side if necessary and identify by block number)*

Ada Programming language, Ada Compiler Validation Summary Report, Ada Compiler Validation Capability, ACVC, Validation Testing, Ada Validation Office, AVO, Ada Validation Facility, AVF, ANSI/MIL-STD-1815A, Ada Joint Program Office, AJPO

20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)*

Ada 86, Version 3.21, SoftTech Inc., National Bureau of Standards, VAX 11/780-11/785 under VAX/VMS, Version 4.7 (Host) to Intel iAPX 80386P under Bare machine (Target) ACVC 1.9.

Ada Compiler Validation Summary Report:

Compiler Name: Ada 86, Version 3.21

Certificate Number: 880708S1.09152

Host:	Target:
VAX 11/780 - 11/785 under VAX/VMS, Version 4.7	Intel iAPX 80386P under Bare machine

Testing Completed July 8, 1988, using ACVC 1.9

This report has been reviewed and is approved.

[illegible]

Ada Validation Facility
Dr. David K. Jefferson
Chief, Information Systems
Engineering Division
National Bureau of Standards
Gaithersburg, MD 20899

[illegible]

Ada Validation Organization
Dr. John F. Kramer
Institute for Defense Analyses
Alexandria, VA 22311

[illegible]

Ada Joint Program Office
Virginia L. Castor
Director
Department of Defense
Washington DC 20301

89 [illegible] 077

AVF Control Number: NBS88VSOF535_6

Ada Compiler
VALIDATION SUMMARY REPORT:

Certificate Number: 880708S1.09152
SoftTech, Inc.
Ada 86, Version 3.21
VAX 11/780 - 11/785 Host and Intel iAPX 80386P Target

Completion of On-Site Testing:
July 8, 1988

Prepared By:
Software Standards Validation Group
Institute for Computer Sciences and Technology
National Bureau of Standards
Building 225, Room A266
Gaithersburg, Maryland 20899

Prepared For:
Ada Joint Program Office
United States Department of Defense
Washington, D.C. 20301-3081

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

This Validation Summary Report ~~(VSR)~~ describes the extent to which a specific Ada compiler conforms to the Ada Standard, ANSI/MIL-STD-1815A. This report explains all technical terms used within it and thoroughly reports the results of testing this compiler using the Ada Compiler Validation Capability ~~(ACVC)~~. An Ada compiler must be implemented according to the Ada Standard, and any implementation-dependent features must conform to the requirements of the Ada Standard. The Ada Standard must be implemented in its entirety, and nothing can be implemented that is not in the Standard.

Even though all validated Ada compilers conform to the Ada Standard, it must be understood that some differences do exist between implementations. The Ada Standard permits some implementation dependencies--for example, the maximum length of identifiers or the maximum values of integer types. Other differences between compilers result from the characteristics of particular operating systems, hardware, or implementation strategies. All the dependencies observed during the process of testing this compiler are given in this report.

This information in this report is derived from the test results produced during validation testing. The validation process includes submitting a suite of standardized tests, the ACVC, as inputs to an Ada compiler and evaluating the results. The purpose of validating is to ensure conformity of the compiler to the Ada Standard by testing that the compiler properly implements legal language constructs and that it identifies and rejects illegal language constructs. The testing also identifies behavior that is implementation dependent but permitted by the Ada Standard. Six classes of test are used. These tests are designed to perform checks at compile time, at link time, and during execution.

1.1 PURPOSE OF THIS VALIDATION SUMMARY REPORT

This VSR documents the results of the validation testing performed on an Ada compiler. Testing was carried out for the following purposes:

> To attempt to identify any language constructs supported by the compiler that do not conform to the Ada Standard
>
> To attempt to identify any unsupported language constructs required by the Ada Standard
>
> To determine that the implementation-dependent behavior is allowed by the Ada Standard

Testing of this compiler was conducted by the National Bureau of Standards according to policies and procedures established by the Ada Validation Organization (AVO). On-site testing was completed July 8, 1988, at SoftTech Corporation, Boston, Mass.

1.2 USE OF THIS VALIDATION SUMMARY REPORT

Consistent with the national laws of the originating country, the AVO may make full and free public disclosure of this report. In the United States, this is provided in accordance with the "Freedom of Information Act" (5 U.S.C. #552). The results of this validation apply only to the computers, operating systems, and compiler versions identified in this report.

The organizations represented on the signature page of this report do not represent or warrant that all statements set forth in this report are accurate and complete, or that the subject compiler has no nonconformities to the Ada Standard other than those presented. Copies of this report are available to the public from:

> Ada Information Clearinghouse
> Ada Joint Program Office
> OUSDRE
> The Pentagon, Rm 3D-139 (Fern Street)
> Washington DC 20301-3081

or from:

> Software Standards Validation Group
> Institute for Computer Sciences and Technology
> National Bureau of Standards
> Building 225, Room A266
> Gaithersburg, Maryland 20899

Questions regarding this report or the validation test results should be directed to the AVF listed above or to:

Ada Validation Organization
Institute for Defense Analyses
1801 North Beauregard Street
Alexandria VA 22311

1.3 REFERENCES

1. Reference Manual for the Ada Programming Language, ANSI/MIL-STD-1815A, February 1983 and ISO 8652-1987.

2. Ada Compiler Validation Procedures and Guidelines. Ada Joint Program Office, 1 January 1987.

3. Ada Compiler Validation Capability Implementers' Guide., December 1986.

1.4 DEFINITION OF TERMS

ACVC	The Ada Compiler Validation Capability. The set of Ada programs that tests the conformity of an Ada compiler to the Ada programming language.
Ada Commentary	An Ada Commentary contains all information relevant to the point addressed by a comment on the Ada Standard. These comments are given a unique identification number having the form AI-ddddd.
Ada Standard	ANSI/MIL-STD-1815A, February 1983 and ISO 8652-1987.
Applicant	The agency requesting validation.
AVF	The Ada Validation Facility. The AVF is responsible for conducting compiler validations according to procedures contained in the Ada Compiler Validation Procedures and Guidelines.
AVO	The Ada Validation Organization. The AVO has oversight authority over all AVF practices for the purpose of maintaining a uniform process for validation of Ada compilers. The AVO provides administrative and technical support for Ada validations to ensure consistent practices.

Compiler	A processor for the Ada language. In the context of this report, a compiler is any language processor, including cross-compilers, translators, and interpreters.
Failed test	An ACVC test for which the compiler generates a result that demonstrates nonconformity to the Ada Standard.
Host	The computer on which the compiler resides.
Inapplicable test	An ACVC test that uses features of the language that a compiler is not required to support or may legitimately support in a way other than the one expected by the test.
Language Maintenance	The Language Maintenance Panel (LMP) is a committee established by the Ada Board to recommend interpretations and Panel possible changes to the ANSI/MIL-STD for Ada.
Passed test	An ACVC test for which a compiler generates the expected result.
Target	The computer for which a compiler generates code.
Test	An Ada program that checks a compiler's conformity regarding a particular feature or a combination of features to the Ada Standard. In the context of this report, the term is used to designate a single test, which may comprise one or more files.
Withdrawn test	An ACVC test found to be incorrect and not used to check conformity to the Ada Standard. A test may be incorrect because it has an invalid test objective, fails to meet its test objective, or contains illegal or erroneous use of the language.

1.5 ACVC TEST CLASSES

Conformity to the Ada Standard is measured using the ACVC. The ACVC contains both legal and illegal Ada programs structured into six test classes: A, B, C, D, E, and L. The first letter of a test name identifies the class to which it belongs. Class A, C, D, and E tests are executable, and special program units are used to report their results during execution. Class B tests are expected to produce compilation errors. Class L tests are expected to produce compilation or link errors.

Class A tests check that legal Ada programs can be successfully compiled and executed. There are no explicit progra components in a Class A test to check semantics. For example, a Class A test checks that reserved words of another language (other than those already reserved in the Ada language) are not treated as reserved words by an Ada compiler. A Class A test is passed if no errors are detected at compile time and the program executes to produce a PASSED message.

Class B tests check that a compiler detects illegal language usage. Class B tests are not executable. Each test in this class is compiled and the resulting compilation listing is examined to verify that every syntax or semantic error in the test is detected. A Class B test is passed if every illegal construct that it contains is detected by the compiler.

Class C tests check that legal Ada programs can be correctly compiled and executed. Each Class C test is self-checking and produces a PASSED, FAILED, or NOT APPLICABLE message indicating the result when it is executed.

Class D tests check the compilation and execution capacities of a compiler. Since there are no capacity requirements placed on a compiler by the Ada Standard for some parameters--for example, the number of identifiers permitted in a compilation or the number of units in a library--a compiler may refuse to compile a Class D test and still be a conforming compiler. Therefore, if a Class D test fails to compile because the capacity of the compiler is exceeded, the test is classified as inapplicable. If a Class D test compiles successfully, it is self-checking and produces a PASSED or FAILED message during execution.

Each Class E test is self-checking and produces a NOT APPLICABLE, PASSED, or FAILED message when it is compiled and executed. However, the Ada Standard permits an implementation to reject programs containing some features addressed by Class E tests during compilation. Therefore, a Class E test is passed by a compiler if it is compiled successfully and executes to produce a PASSED message, or if it is rejected by the compiler for an allowable reason.

Class L tests check that incomplete or illegal Ada programs involving multiple, separately compiled units are detected and not allowed to execute. Class L tests are compiled separately and execution is attempted. A Class L test passes if it is rejected at link time--that is, an attempt to execute the main program must generate an error message before any declarations in the main program or any units referenced by the main program are elaborated.

Two library units, the package REPORT and the procedure CHECK FILE, support the self-checking features of the executable tests. The package REPORT provides the mechanism by which executable tests report PASSED, FAILED, or NOT APPLICABLE results. It also provides a set of identity functions used to defeat some compiler optimizations allowed by the Ada Standard that would circumvent a test objective. The procedure CHECK FILE is used to check the contents of text files written by some of the Class C tests for chapter 14 of the Ada Standard. The operation of

REPORT and CHECK_FILE is checked by a set of executable tests. These tests produce messages that are examined to verify that the units are operating correctly. If these units are not operating correctly, then the validation is not attempted.

The text of the tests in the ACVC follow conventions that are intended to ensure that the tests are reasonably portable without modification. For example, the tests make use of only the basic set of 55 characters, contain lines with a maximum length of 72 characters, use small numeric values, and place features that may not be supported by all implementations in separate tests. However, some tests contain values that require the test to be customized according to implementation-specific values--for example, an illegal file name. A list of the values used for this validation is provided in Appendix C.

A compiler must correctly process each of the tests in the suite and demonstrate conformity to the Ada Standard by either meeting the pass criteria given for the test or by showing that the test is inapplicable to the implementation. The applicability of a test to an implementation is considered each time the implementation is validated. A test that is inapplicable for one validation is not necessarily inapplicable for a subsequent validation. Any test that was determined to contain an illegal language construct or an erroneous language construct is withdrawn from the ACVC and, therefore, is not used in testing a compiler. The tests withdrawn at the time of validation are given in Appendix D.

CHAPTER 2

CONFIGURATION INFORMATION

2.1 CONFIGURATION TESTED

The candidate compilation system for this validation was tested under the following configuration:

Compiler: Ada 86, Version 3.21

ACVC Version: 1.9

Certificate Number:		880708S1.09152
Host Computer:		
	Machine:	VAX 11/780 - 11/785
	Operating System:	VAX/VMS Version 4.7
	Memory Size:	12 megabytes
Target Computer:		
	Machine:	Intel iAPX 80386P
	Operating System:	Bare machine
	Memory Size:	
Communications Network:		DECNET* Ethernet

*DECNET for this implementation represents the use of VAX 11/780-11/785 as host.

2.2 IMPLEMENTATION CHARACTERISTICS

One of the purposes of validating compilers is to determine the behavior of a compiler in those areas of the Ada Standard that permit implementations to differ. Class D and E tests specifically check for such implementation differences. However, tests in other classes also characterize an implementation. The tests demonstrate the following characteristics:

- Capacities.

  The compiler correctly processes tests containing loop statements nested to 65 levels, block statements nested to 65 levels, and recursive procedures separately compiled as subunits nested to 17 levels. It correctly processes a compilation containing 723 variables in the same declarative part. (See test D55A03A..H (8 tests), D56001B, D64005E..G (3 tests), and D29002K.)

- Universal integer calculations.

  An implementation is allowed to reject universal integer calculations having values that exceed SYSTEM.MAX_INT. This implementation 64 bit integer calculations. (See tests D4A002A, D4A002B, D4A004A, and D4A004B.)

- Predefined types.

  This implementation supports the additional predefined types LONG_INTEGER and LONG_FLOAT in the package STANDARD. (See tests B86001BC and B86001D.)

- Based literals.

  An implementation is allowed to reject a based literal with a value exceeding SYSTEM.MAX_INT during compilation, or it may raise NUMERIC_ERROR or CONSTRAINT_ERROR during execution. This implementation raises NUMERIC_ERROR during execution. (See test E24101A.)

- Expression evaluation.

  Apparently all default initialization expressions or record components are evaluated before any value is checked to belong to a component's subtype. (See test C32117A.)

Assignments for subtypes are performed with less precision than the base type. (See test C35712B.)

This implementation uses no extra bits for extra precision. This implementation uses all extra bits for extra range. (See test C35903A.)

Sometimes NUMERIC_ERROR is raised when an integer literal operand in a comparison or membership test is outside the range of the base type. (See test C45232A.)

Apparently NUMERIC_ERROR is raised when a literal operand in a fixed-point comparison or membership test is outside the range of the base type. (See test C45252A.)

Apparently underflow is gradual. (See tests C45524A..Z.)

- Rounding.

The method used for rounding to integer is apparently round to even. (See tests C46012A..Z.)

The method used for rounding to longest integer is apparently round to even. (See tests C46012A..Z.)

The method used for rounding to integer in static universal real expressions is apparently round toward zero. (See test C4A014A.)

- Array types.

An implementation is allowed to raise NUMERIC_ERROR or CONSTRAINT_ERROR for an array having a 'LENGTH that exceeds STANDARD.INTEGER'LAST and/or SYSTEM.MAX_INT. For this implementation:

Declaration of an array type or subtype declaration with more than SYSTEM.MAX_INT components raises NUMERIC_ERROR. (See test C36003A.)

NUMERIC_ERROR is raised when an array type with INTEGER'LAST + 2 components is declared. (See test C36202A.)

NUMERIC_ERROR is raised when an array type with SYSTEM.MAX_INT + 2 components is declared. (See test C36202B.)

A packed BOOLEAN array having a 'LENGTH exceeding INTEGER'LAST raises no exception. (See test C52103X.)

A packed two-dimensional BOOLEAN array with more than

INTEGER'LAST components raises CONSTRAINT_ERROR when the length of a dimension is calculated and exceeds INTEGER'LAST. (See test C52104Y.)

A null array with one dimension of length greater than INTEGER'LAST may raise NUMERIC_ERROR or CONSTRAINT_ERROR either when declared or assigned. Alternatively, an implementation may accept the declaration. However, lengths must match in array slice assignments. This implementation raises CONSTANT_ERROR when array objects are assigned. (See test E52103Y.)

In assigning one-dimensional array types, the expression appears to be evaluated in its entirety before CONSTRAINT_ERROR is raised when checking whether the expression's subtype is compatible with the target's subtype. In assigning two-dimensional array types, the expression does not appear to be evaluated in its entirety before CONSTRAINT_ERROR is raised when checking whether the expression's subtype is compatible with the target's subtype. (See test C52013A.)

- Discriminated types.

During compilation, an implementation is allowed to either accept or reject an incomplete type with discriminants that is used in an access type definition with a compatible discriminant constraint. This implementation accepts such subtype indications during compilation. (See test E38104A.)

In assigning record types with disciminants, the expression appears to be evaluated in its entirety before CONSTRAINT_ERROR is raised when checking whether the expression's subtype is compatible with the target's subtype. (See test C52013A.)

- Aggregates.

In the evaluation of a multi-dimensional aggregate, all choices appear to be evaluated before checking against the index type. (See tests C43207A and C43207B.)

In the evaluation of an aggregate containing subaggregates, all choices are evaluated before being checked for identical bounds. (See test E43212B.)

Not all choices are evaluated before CONSTRAINT_ERROR is raised if a bound in a nonnull range of a nonnull aggregate does not belong to an index subtype. (See test E43211B.)

- Representation clauses.

An implementation might legitimately place restrictions on representation clauses used by some of the tests. If a representation clause is not supported, then the implementation must reject it.

Enumeration representation clauses containing noncontiguous values for enumeration types other than character and boolean types are surported. (See tests C35502I..J, C35502M..N, and A39005F.)

Enumeration representation clauses containing noncontiguous values for character types are supported. (See tests C35507I..J, C35507M..N, and C55B16A.)

Enumeration representation clauses for boolean types containing representational values other than (FALSE => 0, TRUE => 1) are supported. (See tests C35508I..J and C35508M..N.)

Length clauses with SIZE specifications for enumeration types are supported. (See test A39005B.)

Length clauses with STORAGE_SIZE specifications for access types are supported. (See tests A39005C and C87B62B.)

Length clauses with STORAGE_SIZE specifications for task types are supported. (See tests A39005D and C87B62D.)

Length clauses with SMALL specifications are supported. (See tests A39005E and C87B62C.)

Length clauses with SIZE specifications for derived integer types are supported. (See test C87B62A.)

- Pragmas.

The pragma INLINE is supported for procedures. The pragma INLINE is supported for functions. (See tests LA3004A, LA3004B, EA3004C, EA3004D, CA3004E, and CA3004F.)

- Input/output.

The package SEQUENTIAL_IO cannot be instantiated with unconstrained array types and record types with discriminants without defaults. (See tests AE2101C, EE2201D and EE2201E.)

The package DIRECT_IO cannot be instantiated with with unconstrained array types and record types with discriminants without defaults. (See tests AE2101H, EE2401D and EE4201G.)

The director, AJPO, has determined (AI-00332) that every call to

OPEN and CREATE must raise USE_ERROR or NAME_ERROR if file input/output is not supported. This implementation exhibits this behavior for SEQUENTIAL_IO, DIRECT_IO and TEXT_IO.

- Generics.

  Generic subprogram declarations and bodies can compiled in separate compilations. (See tests CA1012A and CA2009F.)

  Generic package declarations and bodies can be compiled in separate compilations. (See tests CA2009C, BC3204C, and BC3205D.)

  Generic unit bodies and their subunits can be compiled in separate compilations. (See test CA3011A.)

CHAPTER 3

TEST INFORMATION

SULTS

of the ACVC comprises 3122 tests. When this compiler was
ests had been withdrawn because of test errors. The AVF
hat 412 tests were inapplicable to this implementation. All
e tests were processed during validation testing.
s to the code, processing, or grading for 25 tests were
successfully demonstrate the test objective. (See section

ludes that the testing results demonstrate acceptable
o the Ada Standard.

OF TEST RESULTS BY CLASS

RESULT	TEST CLASS						TOTAL
	A	B	C	D	E	L	
Passed	105	1048	1454	17	12	46	2682
Inapplicable	5	3	399	0	5	0	412
Withdrawn	3	2	21	0	2	0	28
TOTAL	113	1053	1874	17	19	46	3122

3.3 SUMMARY OF TEST RESULTS BY CHAPTER

RESULT	CHAPTER													TOTAL
	2	3	4	5	6	7	8	9	10	11	12	13	14	
Passed	190	498	535	245	165	98	141	327	137	36	234	3	73	2682
Inapplicable	14	74	139	3	0	0	2	0	0	0	0	0	180	412
Withdrawn	2	14	3	0	1	1	2	0	0	0	2	1	2	28
TOTAL	206	586	677	248	166	99	145	327	137	36	236	4	255	3122

3.4 WITHDRAWN TESTS

The following 28 tests were withdrawn from ACVC Version 1.9 at the time of this validation:

B28003A	E28005C	C34004A	C35502P	A35902C	C35904A
C35904B	C35A03E	C35A03R	C37213H	C37213J	C37215C
C37215E	C37215G	C37215H	C38102C	C41402A	C45332A
C45614C	E66001D	A74106C	C85018B	C87B04B	CC1311B
BC3105A	AD1A01A	CE2401H	CE3208A		

See Appendix D for the reason that each of these tests was withdrawn.

3.5 INAPPLICABLE TESTS

Some tests do not apply to all compilers because they make use of features that a compiler is not required by the Ada Standard to support. Others may depend on the result of another test that is either inapplicable or withdrawn. The applicability of a test to an implementation is considered each time a validation is attempted. A test that is inapplicable for one validation attempt is not necessarily inapplicable for a subsequent attempt. For this validation attempt, 412 test were inapplicable for the reasons indicated:

C35702A uses SHORT_FLOAT which is not supported by this implementation.

A35801E At the case statement (lines 54-63), the optimizer tries to identify which of the cases will be done during execution. The optimizer recognizes that the variable "I" which is of type integer, is not initialized and appropriately raises a PROGRAM_ERROR exception. NOTE: This test passes without the /OPTIMIZE option.

A39005G uses a record representation clause which is not supported by this compiler.

The following (14) tests use SHORT_INTEGER, which is not supported by this compiler.

C45231B	C45304B	C45502B	C45503B	C45504B
C45504E	C45611B	C45613B	C45614B	C45631B
C45632B	B52004E	C55B07B	B55B09D	

C45231D requires a macro substitution for any predefined numeric types other than INTEGER, SHORT_INTEGER, LONG_INTEGER, FLOAT, SHORT_FLOAT, and LONG_FLOAT. This compiler does not support any such types.

C45304A, C45304C and C46014A expect exceptions to be raised as the result of performing "dead assignments" (assignments to a variable whose value is never used in the program).

C45531M, C45531N, C45532M, and C45532N use fine 48-bit fixed-point base types which are not supported by this compiler.

C45531O, C45531P, C45532O, and C45532P use coarse 48-bit fixed-point base types which are not supported by this compiler.

B86001D requires a predefined numeric type other than those defined by the Ada language in package STANDARD. There is no such type for this implementation.

C86001F redefines package SYSTEM, but TEXT_IO is made obsolete by this new definition in this implementation and the test cannot be executed since the package REPORT is dependent on the package TEXT_IO.

AE2101C, EE2201D, and EE2201E use instantiations of package SEQUENTIAL_IO with unconstrained array types and record types having discriminants without defaults. These instantiations are rejected by this compiler.

AE2101H, EE2401D, and EE2401G use instantiations of package DIRECT_IO with unconstrained array types and record types having discriminants without defaults. These instantiations are rejected by this compiler.

The following 174 tests are inapplicable because sequential, text, and direct access files are not supported.

CE2102C	CE2102G..H(2)	CE2102K	CE2104A..D(4)
CE2105A..B(2)	CE2106A..B(2)	CE2107A..I(9)	CE2108A..D(4)
CE2109A..C(3)	CE2110A..C(3)	CE2111A..E(5)	CE2111G..H(2)
CE2115A..B(2)	CE2201A..C(3)	CE2201F..G(2)	CE2204A..B(2)
CE2208B	CE2210A	CE2401A..C(3)	CE2401E..F(2)

CE2404A	CE2405B	CE2406A	CE2407A
CE2408A	CE2409A	CE2410A	CE2411A
AE3101A	CE3102B	EE3102C	CE3103A
CE3104A	CE3107A	CE3108A.B(2)	CE3109A
CE3110A	CE3111A..E(5)	CE3112A..B(2)	CE3114A..B(2)
CE3115A	CE3203A	CE3301A..C(3)	CE3302A
CE3305A	CE3402A..D(4)	CE3403A..C(3)	CE3403E..F(2)
CE3404A..C(3)	CE3405A..D(4)	CE3406A..D(4)	CE3407A..C(3)
CE3408A..C(3)	CE3409A	CE3409C..F(4)	CE3410A
CE3410C..F(4)	CE3411A	CE3412A	CE3413A
CE3413C	CE3602A..D(4)	CE3603A	CE3604A
CE3605A..E(5)	CE3606A..B(2)	CE3704A..B(2)	CE3704D..F(3)
CE3704M..O(3)	CE3706D	CE3706F	CE3804A..E(5)
CE3804G	CE3804I	CE3804K	CE3804M
CE3805A..B(2)	CE3806A	CE3806D..E(2)	CE3905A..C(3)
CE3905L	CE3906A..C(3)	CE3906E..F(2)	

The following 201 tests require a floating-point accuracy that exceeds the maximum of 15 digits supported by this implementation:

C24113L..Y (14 tests)	C35705L..Y (14 tests)
C35706L..Y (14 tests)	C35707L..Y (14 tests)
C35708L..Y (14 tests)	C35802L..Z (15 tests)
C45241L..Y (14 tests)	C45321L..Y (14 tests)
C45421L..Y (14 tests)	C45521L..Z (15 tests)
C45524L..Z (15 tests)	C45621L..Z (15 tests)
C45641L..Y (14 tests)	C46012L..Z (15 tests)

3.6 TEST, PROCESSING, AND EVALUATION MODIFICATIONS

It is expected that some tests will require modifications of code, processing, or evaluation in order to compensate for legitimate implementation behavior. Modifications are made by the AVF in cases where legitimate implementation behavior prevents the successful completion of an (otherwise) applicable test. Examples of such modifications include: adding a length clause to alter the default size of a collection; splitting a Class B test into sub-tests so that all errors are detected; and confirming that messages produced by an executable test demonstrate conforming behavior that wasn't anticipated by the test (such as raising one exception instead of another).

Modifications were required for 24 Class B tests.

The following Class B tests were split because syntax errors at one point resulted in the compiler not detecting other errors in the test:

B2A003A..C (3 tests)		B33201C	B33202C	B33203C
B33301C	B37106A	B37201A	B37301I	B37307B
B38001C	B38003A..B	B38009A..B	B44001A	B51001A
B54A01C	B54A01L	B95063A	BC1008A	BC1201L
BC3013A				

C4A012B requires that a CONSTRAINT_ERROR be raised in a context where a NUMERIC_ERROR is relivant on line 35, etc. The test has been evaluated and recommended to be graded as passed.

3.7 ADDITIONAL TESTING INFORMATION

3.7.1 Prevalidation

Prior to validation, a set of test results for ACVC Version 1.9 produced by the Ada 86 was submitted to the AVF by the applicant for review. Analysis of these results demonstrated that the compiler successfully passed all applicable tests, and the compiler exhibited the expected behavior on all inapplicable tests.

3.7.2 Test Method

Testing of the Ada 86 using ACVC Version 1.9 was conducted on-site by a validation team from the AVF. The configuration consisted of a VAX 11/780 - 11/785 host operating under VAX/VMS, Version 4.7, and an iAPX 80386P target operating under bare machine. The host and target computers were linked via DECNET.

A magnetic tape containing all tests was taken on-site by the validation team for processing. Tests that make use of implementation-specific values were customized on-site after the magnetic tape was loaded. Tests requiring modifications during the prevalidation testing were not included in their modified form on the magnetic tape. The contents of the magnetic tape were loaded directly onto the host computer.

After the test files were loaded to disk, the full set of tests was compiled and linked on the VAX 11/780 - 11/785, and all executable tests were run on the iAPX 80386P. Object files were linked on the host computer, and executable images were transferred to the target computer via DECNET. Results were printed from the host computer, with results being transferred to the host computer via DECNET.

The compiler was tested using command scripts provided by SofTech, Incorporated and reviewed by the validation team. The compiler was tested using all default option settings without exception.

Tests were compiled, linked, and executed (as appropriate) using a single host computer and a single target computer. Test output, compilation listings, and job logs were captured on magnetic tape and archived at the AVF. The listings examined on-site by the validation team were also archived.

3.7.3 Test Site

Testing was conducted at SofTech, Incorporated, Boston, Massachusetts and was completed on July 8, 1988.

DECLARATION OF CONFORMANCE

APPENDIX A

DECLARATION OF CONFORMANCE

Compiler Implementer:	SofTech Inc. 460 Totten Pond Road Waltham, MA 02254
Ada Validation Facility:	National Bureau of Standards (NBS) Institute for Computer Sciences and Technology (ICST) Software Standards Validation Group Building 225, Room A266 Gaithersburg, MD 20899-9999

Ada Compiler Validation Capability (ACVC) Version: 1.9

BASE CONFIGURATION(S)

Base Compiler Name:	Ada86	Version:	3.21
Host Architecture	- ISA: VAX 11/780 - 11/785	OS&VER #:	VAX/VMS 4.7
Target Architecture	- ISA: Intel iAPX 8086	OS&VER #:	(bare machine)
Base Compiler Name:	Ada86	Version:	3.21
Host Architecture	- ISA: VAX 11/780 - 11/785	OS&VER #:	VAX/VMS 4.7
Target Architecture	- ISA: Intel iAPX 80186	OS&VER #:	(bare machine)
Base Compiler Name:	Ada86	Version:	3.21
Host Architecture	- ISA: VAX 11/780 - 11/785	OS&VER #:	VAX/VMS 4.7
Target Architecture	- ISA: Intel iAPX 80286 real mode	OS&VER #:	(bare machine)
Base Compiler Name:	Ada86	Version:	3.21
Host Architecture	- ISA: VAX 11/780 - 11/785	OS&VER #:	VAX/VMS 4.7
Target Architecture	- ISA: Intel iAPX 80286 protected mode	OS&VER #:	(bare machine)
Base Compiler Name.	Ada86	Version:	3.21
Host Architecture	- ISA: VAX 11/780 - 11/785	OS&VER #:	VAX/VMS 4.7
Target Architecture	- ISA: Intel iAPX 80386 compatible real mode	OS&VER #:	(bare machine)
Base Compiler Name:	Ada86	Version:	3.21
Host Architecture	- ISA: VAX 11/780 - 11/785	OS&VER #:	VAX/VMS 4.7
Target Architecture	- ISA: Intel iAPX 80386 compatible protected mode	OS&VER #:	(bare machine)

DERIVED COMPILER REGISTRATION
EQUIVALENT CONFIGURATION(S)

Base Compiler Name:	Ada86	Version: 3.21, 1.59, 1.70
Host Architecture _	- ISA: VAX 700 and 8000 Series	OS&VER #: VAX/VMS 4.7
Target Architecture	- ISA: Intel iAPX 8086	OS&VER #: (bare machine)
Target Architecture	- ISA: Intel iAPX 80186	OS&VER #: (bare machine)
Target Architecture	- ISA: Intel iAPX 80286 real mode	OS&VER #: (bare machine)
Target Architecture	- ISA: Intel iAPX 80286 protected	OS&VER #: (bare machine)
Target Architecture	- ISA: Intel iAPX 80386 comp real	OS&VER #: (bare machine)
Target Architecture	- ISA: Intel iAPX 80386 comp prot	OS&VER #: (bare machine)

Base Compiler Name:	Ada86	Version: 3.21, 1.59, 1.70
Host Architecture	- ISA: MicroVAX II	OS&VER #: MicroVMS 4.7
Target Architecture	- ISA: Intel iAPX 8086	OS&VER #: (bare machine)
Target Architecture	- ISA: Intel iAPX 80186	OS&VER #: (bare machine)
Target Architecture	- ISA: Intel iAPX 80286 real mode	OS&VER #: (bare machine)
Target Architecture	- ISA: Intel iAPX 80286 protected	OS&VER #: (bare machine)
Target Architecture	- ISA: Intel iAPX 80386 comp real	OS&VER #: (bare machine)
Target Architecture	- ISA: Intel iAPX 80386 comp prot	OS&VER #: (bare machine)

Implementer's Declaration

I, the undersigned, representing SofTech, Inc., have implemented no deliberate extensions to the Ada Language Standard ANSI/MIL-STD-1815A in the compiler(s) listed in this declaration. I declare that the SofTech Inc. is _ the owner on record of the Ada language compiler(s) listed above and, as such, is responsible for maintaining said compiler(s) in conformance to ANSI-MIL-STD-1815A. All certificates and registrations for Ada language compiler(s) listed in this declaration shall be made only in the owner's corporate name.

[illegible], General Manager July 29, 1988

Implementer's Signature and Title Date

Implementer's Declaration

Owner's Declaration

I, the undersigned, representing SofTech Inc., take full responsibility for implementation and maintenance of the Ada compiler(s) listed above, and agree to the public disclosure of the final Validation Summary Report. I further agree to continue to comply with the Ada trademark policy, as defined by the Ada Joint Program Office. I declare that all of the Ada language compilers listed, and their host/target performance are in compliance with the Ada Language Standard ANSI/MIL-STD-1815A. I have reviewed the Validation Summary Report for the compilers(s) and concur with the contents.

[illegible], President 8/1/88

Owner's Signature and Title Date

APPENDIX B

APPENDIX F OF THE Ada STANDARD

The only allowed implementation dependencies correspond to implementation-dependent pragmas, to certain machine-dependent conventions as mentioned in chapter 13 of the Ada Standard, and to certain allowed restrictions on representation clauses. The implementation-dependent characteristics of the Ada 86, Version 3.21, are described in the following sections which discuss topics in Appendix F of the Ada Standard. Implementation- specific portions of the package STANDARD are also included in this appendix.

```
package STANDARD is


     type INTEGER is range -32_768 .. 32_767;

     type LONG_INTEGER is range -2_147_483_648 .. 2_147_483_647;

     type FLOAT is digits 6 range -(2#1.111_(5)1111#E+127) ..
                                   (2#1.111_(5)1111#E+127);

     type LONG_FLOAT is digits 15 range
                              -(2#1.111_(12)1111_1#E+1023 ..
                                (2#1.111_(12)1111_1#E+1023;

     type DURATION is delta 2.0**(-14) range -131_072.0 ..
                                               131_072.0;


end STANDARD;
```

APPENDIX F

APPENDIX F OF THE Ada STANDARD for SofTech's Ada86 toolset

The only allowed implementation dependencies correspond to implementation-dependent pragmas, to certain machine_dependent conventions as mentioned in chapter 13 of MIL-STD-1815A, and to certain allowed restrictions on representation clauses. The implementation-dependent characteristics are described in the following sections which discuss topics one through eight as stated in Appendix F of the Ada Language Reference Manual (ANSI/MIL-STD-1815A). Two other sections, package STANDARD and file naming conventions, are also included in this appendix.

vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
(1) Implementation-Dependent Pragmas
^^^

++
This section may be copied from the applicant's documentation, but make sure it covers all the items below.
++

The pragmas described below are implementation-defined.

Pragma TITLE (arg);

This is a listing control pragma. "Arg" is a CHARACTER string literal that is to appear on the second line of each page of every listing produced for a compilation unit in the compilation. At most, one such pragma may appear for any compilation, and it must be the first unit in the compilation (comments and other pragmas excepted).

For many real time applications, fast software reaction to hardware interrupts is important. A group of pragmas is provided in recognition of this requirement.

If an Ada task entry has been equated to a hardware interrupt through an address clause (c.f. LRM 13.5.1), the occurrence of the hardware interrupt in question is interpreted by the RSL as an entry call to the corresponding task entry. The object code generated to implement interrupt entries includes some overhead, since the Ada programmer is allowed to make use of the full Ada language within the accept body for the interrupt entry.

The pragmas described below let the user specify that interrupt entries, and the tasks that contain them, meet certain restrictions. The restrictions speed up the software response to hardware interrupts.

Pragma FAST_INTERRUPT_ENTRY (entry_simple_name,
                              SYSTEM.ENTRY_KIND literal)

This pragma specifies that the named task entry has only accept bodies that execute completely with (maskable) interrupts disabled, and that none of these accept bodies performs operations that may potentially lead to task switches away from the accept body.

Pragma INTERRUPT_HANDLER_TASK

This pragma specifies that the task at hand is degenerate in that the whole task body consits of a single loop, which in turn contains one or several accept statements for fast interrupt entries, and which accesses only global variables.

Pragma TRIVIAL_ENTRY (entry_simple_name)

This pragma specifies that all accept statements for the named entry are degenerate in that their sequence of statements is empty. Moreover, all entry calls to such an entry are conditional entry calls, and they are issued only from within accept bodies for fast interrupt entries.

vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
(2) Implementation-Dependent Attributes
^^^

The predefined attribute, X'DISP, is not supported.

```
vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
(3)   Package SYSTEM
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

--#  Copyright 1986 Softech,  Inc., all rights reserved.

-- Copyright (C) 1987, SofTech, Inc.

package SYSTEM is  --[LRM 13.7 and F]


  type    WORD is range 0..16#FFFF#;
  for     WORD'SIZE use 16;                        --see[ LRM 3.4(10) ]
          -- Ada SIZE attribute gives 16, but machine size is 32.

  type    BYTE is range 0..255;
  for     BYTE'SIZE use  8;
          -- Ada SIZE attribute gives 8, but machine size is 16.

  subtype REGISTER          is SYSTEM.WORD;

  --#START iAPX86, iAPX186, iAPX286R, iAPX386R, PC_DOS
      subtype SEGMENT_REGISTER is SYSTEM.REGISTER;

      NULL_SEGMENT: constant SYSTEM.SEGMENT_REGISTER := 0;
  --#STOP iAPX86, iAPX186, iAPX286R, iAPX386R, PC_DOS

   --#START iAPX286P, iAPX386P

--#    type SEGMENT_LENGTH_IN_BYTES is range 1..65536;
--#       -- Gives the range the length an iAPX286 memory segment can be.
--#       -- The hardware deals with segment limits which is the length
--#       -- of the segment relative to the base minus one.
--#       -- It is more convenient to use the length of the segment
--#       -- so this type is provided.
--#       -- See page 7-13 of the Intel iAPX286 Programmer's Reference Manual.

--#    type PRIVILEGE_LEVEL is range 0..3;
--#    for PRIVILEGE_LEVEL'SIZE use 2;
--#       -- Privilege level as defined by the iAPX286 hardware.

--#    -- The following types form an iAPX286 selector as described on page 7-11
--#    -- of the Intel iAPX286 Programmer's Reference Manual.

--#    type DESCRIPTOR_TABLE_INDEX is range 0..8191;
--#    for DESCRIPTOR_TABLE_INDEX'SIZE use 13;
--#       -- Index into the global or local descriptor table.

--#    type DESCRIPTOR_TABLE_INDICATOR is
--#       (USE_GLOBAL_DESCRIPTOR_TABLE, USE_LOCAL_DESCRIPTOR_TABLE);

--#    for DESCRIPTOR_TABLE_INDICATOR use
--#       (USE_GLOBAL_DESCRIPTOR_TABLE => 0, USE_LOCAL_DESCRIPTOR_TABLE => 1);

--#    for DESCRIPTOR_TABLE_INDICATOR'SIZE use 1;
--#       -- Indicates whether to use the global or the local descriptor table.

--#    type SEGMENT_REGISTER is
--#       record
--#          -- This is a segment selector as defined by the iAPX286 hardware.
--#         -- See page 7-11 of the Intel iAPX286 Programmer's Reference Manual.

--#         DESCRIPTOR_INDEX: DESCRIPTOR_TABLE_INDEX;
--#            -- This is an index into either the global or the local
--#            -- descriptor table.  The index will select one of the 8 byte
--#            -- descriptors in the table.
--#            -- The table to use is given by the TABLE_INDICATOR field.
--#            -- NOTE:
--#            -- Even if an index is in the proper range, it might not refer
--#            -- to an existing or valid descriptor.  See page 7-5 of the
--#            -- Intel iAPX286 Programmer's Reference Manual.

--#         TABLE_INDICATOR: DESCRIPTOR_TABLE_INDICATOR;
--#            -- Whether the index is an index into the global or the local
--#            -- descriptor table;

--#         REQUESTED_PRIVILEGE_LEVEL: PRIVILEGE_LEVEL;
--#            -- The requested privilege level reflects the privilege level of
--#            -- original supplier of the selector.  Needed when addresses are
--#            -- passed through intermediate levels.  See page 7-14 of the
--#            -- Intel iAPX286 Programmer's Reference Manual.
--#      end record;

--#   for SEGMENT_REGISTER'SIZE use 16;

--#   for SEGMENT_REGISTER use
--#      record
--#         REQUESTED_PRIVILEGE_LEVEL at 0 range 0..1;
--#         TABLE_INDICATOR           at 0 range 2..2;
--#         DESCRIPTOR_INDEX          at 0 range 3..15;
--#      end record;

--#   NULL_SEGMENT   : constant SYSTEM.SEGMENT_REGISTER :=
--#      (0, USE_GLOBAL_DESCRIPTOR_TABLE, 0);

--#   -- Index of the IDT descriptor in GDT
--#   IDT_INDEX      : constant DESCRIPTOR_TABLE_INDEX := 2;

--#   -- Size in bytes of the descriptors in IDT
--#   IDT_ENTRY_SIZE : constant := 8;

   --#STOP iAPX286P, iAPX386P

  subtype OFFSET_REGISTER  is SYSTEM.REGISTER;

  type    ADDRESS is
    record
      SEGMENT: SYSTEM.SEGMENT_REGISTER;
      OFFSET : SYSTEM.OFFSET_REGISTER;
    end record;

  for     ADDRESS'SIZE use 32;

  for     ADDRESS use                             --see[ UM83 4-10, ASM86 6-57,
    record                                        --     Ada Issue  7]
      OFFSET  at 0 range 0..15;
      SEGMENT at 2 range 0..15;
    end record;

  --#START iAPX86, iAPX186, iAPX286R, iAPX386R, PC_DOS
```

```
        NULL_ADDRESS : constant SYSTEM.ADDRESS := ( 0, 0 );
    --#STOP iAPX86, iAPX186, iAPX286R, iAPX386R, PC_DOS

    --#START iAPX286P, iAPX386P
  --#  NULL_ADDRESS : constant SYSTEM.ADDRESS := (SYSTEM.NULL_SEGMENT, 0);
    --#STOP iAPX286P, iAPX386P


    subtype IO_ADDRESS        is SYSTEM.REGISTER;

    --#START iAPX86, iAPX186, iAPX286R, iAPX386R, PC_DOS
        type    ABSOLUTE_ADDRESS is range 0..16#FFFFF#;
        for     ABSOLUTE_ADDRESS'SIZE use 20;
            -- Ada SIZE attribute gives 20, but machine size is 32.
    --#STOP iAPX86, iAPX186, iAPX286R, iAPX386R, PC_DOS

     --#START iAPX286P, iAPX386P
  --#   type    ABSOLUTE_ADDRESS is range 0..16#FFFFFF#;
  --#   for     ABSOLUTE_ADDRESS'SIZE use 24;
  --#           -- Ada SIZE attribute gives 24, but machine size is 32.
     --#STOP iAPX286P, iAPX386P

    type    NAME is ( VAX780_VMS, iAPX86, iAPX186, iAPX286R, iAPX286P,
                      PC_DOS, iAPX386R, iAPX386P );

    --#START iAPX86
        SYSTEM_NAME : constant SYSTEM.NAME := ( SYSTEM.iAPX86 );
       --Intel 8086 in real address mode.
    --#STOP iAPX86

    --#START iAPX186
  --#   SYSTEM_NAME : constant SYSTEM.NAME := ( SYSTEM.iAPX186 );
  --#  --Intel 80186 in real address mode.
    --#STOP iAPX186

    --#START iAPX286R
  --#   SYSTEM_NAME : constant SYSTEM.NAME := ( SYSTEM.iAPX286R );
  --#  --Intel 80286 in real address mode.
    --#STOP iAPX286R, iAPX386R

    --#START iAPX286P
  --#   SYSTEM_NAME : constant SYSTEM.NAME := ( SYSTEM.iAPX286P );
  --#  --Intel 80286 in protected virtual address mode.
    --#STOP iAPX286P

    --#START iAPX386R
  --#   SYSTEM_NAME : constant SYSTEM.NAME := ( SYSTEM.iAPX386R );
  --#  --Intel 80386 in real address mode.
    --#STOP iAPX386R

    --#START iAPX386P
  --#   SYSTEM_NAME : constant SYSTEM.NAME := ( SYSTEM.iAPX386P );
  --#  --Intel 80386 in protected virtual address mode (iAPX286P subset).
    --#STOP iAPX386P

    --#START PC_DOS
  --#   SYSTEM_NAME : constant SYSTEM.NAME := ( SYSTEM.PC_DOS );
  --#  --Intel 8086 in real address mode.
    --#STOP PC_DOS
```

```
STORAGE_UNIT: constant := 8;

--#START iAPX86, iAPX186, iAPX286R, iAPX386R, PC_DOS
   MEMORY_SIZE : constant := (2**20)-1 ;        -- 1_048_575
--#STOP  iAPX86, iAPX186, iAPX286R, iAPX386R, PC_DOS

 --#START iAPX286P, iAPX386P
--#  MEMORY_SIZE :_constant := (2**24)-1 ;        --  16_777_215
 --#STOP iAPX286P, iAPX386P

MIN_INT      : constant := -(2**31)  ;       --  -2_147_483_648
MAX_INT      : constant :=  (2**31)-1  ;     --   2_147_483_647


MAX_DIGITS   : constant := 15;               --Changed from 9 to 15 to match
                                             --change to LONG_FLOAT in package
                                             --STANDARD
  --Note that the Intel 8087 Numeric Data Processor HAS dictated the
  --value of MAX_DIGITS.


MAX_MANTISSA: constant := 31;
FINE_DELTA   : constant := 4.656_612_873_077_392_578_125E-10;  -- 2.0**(-31);


type INTERRUPT_TYPE_NUMBER is range 0..255;


--Interrupts having the following Interrupt Type Numbers are specific to the
--iAPX86, iAPX186, and iAPX286 CPUs:
--(Note that the following are declared as CONSTANT universal integers rather
--than CONSTANT SYSTEM.INTERRUPT_TYPE_NUMBERs.  This is so that they can be
--used in MACHINE_CODE statements, which require all expressions to be static.
--At least in our implementation, conversions such as
--"MACHINE_CODE.BYTE_VAL( SYSTEM.DISPATCH_CODE_INTERRUPT )" are not considered
--to be static.

DIVIDE_ERROR_INTERRUPT                  : constant := 0;
  --Ada semantics dictate that this interrupt must be interpreted as the
  --exception NUMERIC_ERROR.

SINGLE_STEP_INTERRUPT                   : constant :=  1;
  --The non-maskable internal interrupt generated by the CPU after the
  --execution of an instruction when the Trap Flag (TF) is set.

NON_MASKABLE_INTERRUPT                  : constant :=  2;
  --The hardware-generated external interrupt delivered to the CPU via the
  --NMI pin.  This interrupt can never be disabled by software and can
  --penetrate critical regions.

OVERFLOW_INTERRUPT                      : constant :=  4;
  --Ada semantics dictate that this interrupt must be interpreted as the
  --exception NUMERIC_ERROR.


--Interrupts having the following Interrupt Type Numbers are specific to the
--actual configuration of the iSBC 86/30 board rather than just its CPU:

--#START iAPX86, iAPX286R, iAPX386R, iAPX286P, iAPX386P
```

```
       RSL_CLOCK_INTERRUPT                        : constant :=  64;
    --#STOP  iAPX86, iAPX286R, iAPX386R, iAPX286P, iAPX386P
    --#START PC_DOS
  --#    RSL_CLOCK_INTERRUPT                        : constant :=  8;
    --#STOP  PC_DOS
    --#START iAPX86, iAPX286R, iAPX386R, iAPX286P, iAPX386P, PC_DOS
      --This interrupt is reserved for the use of the RSL in maintaining the
      --real-time clock and for the support of DELAY statements.
      --
    --#STOP  iAPX86, iAPX286R, iAPX386R, iAPX286P, iAPX386P, PC_DOS
    --#START iAPX86, iAPX286R, iAPX386R, iAPX286P, iAPX386P
      --Upper 5 bits, supplied by PIC, are 2#01000#,
    --#STOP  iAPX86, iAPX286R, iAPX386R, iAPX286P, iAPX386P
    --#START PC_DOS
  --# --Upper 5 bits, supplied by PIC, are 2#00001#,
    --#STOP  PC_DOS
    --#START iAPX86, iAPX286R, iAPX386R, iAPX286P, iAPX386P, PC_DOS
      --lower 3 bits, derived from PIC input number (IR0), are 2#000#.
      --
      --By default, this interrupt is the highest in priority.
      --
      --Assumption: The OUT0 output of the PIT (alias "TIMER 0 INTR") is
      --connected to the PIC input IR0.
    --#STOP  iAPX86, iAPX286R, iAPX386R, iAPX286P, iAPX386P, PC_DOS

      --#START iAPX186
  --#    RSL_CLOCK_INTERRUPT                        : constant := 18;
  --# --This interrupt is reserved for the use of the RSL in maintaining the
  --# --real-time clock.

  --# DELAY_EXPIRY_INTERRUPT           : constant := 8;
  --# --This interrupt is reserved for the use of the RSL in implementing delays
  --# --of less than a full RSL clock cycle.
    --#STOP  iAPX186

    --#START iAPX86
       NUMERIC_PROCESSOR_INTERRUPT                : constant := 71;
      --This interrupt must be interpreted as the exception NUMERIC_ERROR.
      --
      --Upper 5 bits, supplied by PIC, are 2#01000#,
      --lower 3 bits, derived from PIC input number (IR7), are 2#111#.
      --
      --By default, this interrupt is the lowest in priority.
      --
      --Assumption: The 8087 interrupt line (alias Math Interrupt or "MINT"), is
      --connected to the PIC input IR7.
    --#STOP iAPX86

    --#START PC_DOS
  --#    NUMERIC_PROCESSOR_INTERRUPT                : constant := NON_MASKABLE_INTERRUP
  T;
  --# -- This interrupt must be interpreted as the exception NUMERIC_ERROR
  --# -- When bits 6 and 7 of port 16#00C2# are zero.  Otherwise it indicates
  --# -- an I/O Channel Check or a Read/Write Memory Parity Check.
  --# -- The IBM-PC delivers the numeric processor exceptions via the
  --# -- non-maskable interrupt.
  --# --
    --#STOP  PC_DOS

    --#START iAPX186
```

```
--#   NUMERIC_PROCESSOR_INTERRUPT            : constant := 15;
--# --This interrupt must be interpreted as the exception NUMERIC_ERROR.
--# --
--# --Upper 5 bits, supplied by PIC, are 2#00001#,
--# --lower 3 bits, derived from PIC input number (IR7), are 2#111#.
--# --
--# --By default, this interrupt is the lowest in priority.
--# --
--# --Assumption: The 8087 interrupt line (alias Math Interrupt or "MINT"), is
--# --connected to the PIC input IR7.
  --#STOP iAPX186

  --#START iAPX286R, iAPX386R, iAPX286P, iAPX386P
--#   NUMERIC_PROCESSOR_INTERRUPT            : constant := 16;
--#      --alias Processor Extension Error [PRM Numeric Supplement 1-37]
  --#STOP  iAPX286R, iAPX386R, iAPX286P, iAPX386P

--*** The following RSL internal interrupt type numbers must be changed
--    when the compiler interface has been changed.

--#START iAPX86, iAPX186, iAPX286R, iAPX386R, iAPX286P, iAPX386P

      --The software interrupt having the following Interrupt Type Number is use
d
      --internally and exclusively by the RSL to check if the current stack
      --has enough space:

      CHECK_STACK_INTERRUPT                  : constant := 48;

      --The software interrupt having the following Interrupt Type Number is use
d

      --internally and exclusively by the RSL to effect switching between tasks:

      DISPATCH_CODE_INTERRUPT                : constant := 32;


      --Interrupts having the following Interrupt Type Numbers (all
      --software-generated) are used internally and exclusively by the generated
      --code for effecting subprogram entry sequences where there is no SFDD:

      ENTER_SUBPROGRAM_WITHOUT_LPP_INTERRUPT : constant := 49;
    --The generated code uses this interrupt to effect a subprogram entry
    --sequence without a Lexical Parent Pointer.

      ENTER_SUBPROGRAM_INTERRUPT             : constant := 50;
    --The generated code uses this interrupt to effect a subprogram entry
    --sequence with a Lexical Parent Pointer.

      --Interrupts having the following Interrupt Type Numbers (all software-
      --generated) are used internally and exclusively by the generated code to
      --cause certain Ada exceptions to be forced:

      PROGRAM_ERROR_INTERRUPT                : constant := 53;
    --This interrupt mus  be interpreted as the exception PROGRAM_ERROR.

      CONSTRAINT_ERROR_INTERRUPT             : constant := 54;
    --This interrupt must be interpreted as the exception CONSTRAINT_ERROR.

      NUMERIC_ERROR_INTERRUPT                : constant := 55;
    --This interrupt must be interpreted as the exception NUMERIC_ERROR.
    --
    --
    -- --Interrupts having the following Interrupt Type Numbers (all software-
    -- --generated) are used internally and exclusively by the generated code to
    -- --cause certain RSL services to be invoked:
    --
    - ALLOCATE_OBJECT_INTERRUPT                : constant := 56;
    --This interrupt causes an object to be allocated in the heap of the
    --anonymous task.


      --The software interrupts having the following Interrupt Type Numbers are
used
      --internally and exclusively by the RSL to effect entry to and exit from
      --Innocuous Critical Regions:

      ENTER_INNOCUOUS_CRITICAL_REGION_INTERRUPT: constant := 33;


      LEAVE_INNOCUOUS_CRITICAL_REGION_INTERRUPT: constant := 34;

      --The software interrupts having the following Interrupt Type Numbers are
      --defined (and used) by the RSL and can be used by the user:
      --
      -- Used to halt the execution of the program from any point.
      HALT_INTERRUPT                : constant := 36;
      END_OF_PROGRAM_INTERRUPT      : constant := 37;

      STORAGE_ERROR_INTERRUPT       : constant := 38;
        --This interrupt must be interpreted as the exception STORAGE_ERROR.
--#STOP iAPX86, iAPX186, iAPX286R, iAPX386R, iAPX286P, iAPX386P

--   --#START iAPX286P, iAPX386P
--   LOAD_TASK_REGISTER_INTERRUPT : constant := 37;
--   CLEAR_TS_FLAG_INTERRUPT      : constant := 38;
--   HALT_INTERRUPT               : constant := 39;
--   --#STOP  iAPX286P, iAPX386P


  --Interrupts having the following Interrupt Type Numbers are specific to the
  --Intel iAPX 186 and iAPX 286 CPUs:

  BOUND_EXCEPTION_INTERRUPT                : constant :=  5;
    --This interrupt will be interpreted as the exception CONSTRAINT_ERROR.

  UNDEFINED_OPCODE_EXCEPTION_INTERRUPT     : constant :=  6;
    --This interrupt will be interpreted as the exception PROGRAM_ERROR.

  PROCESSOR_EXTENSION_NOT_AVAILABLE_INTERRUPT: constant :=  7;
    --This interrupt will be interpreted as the exception PROGRAM_ERROR.
```

```
--#START PC_DOS

--#   --The software interrupt having the following Interrupt Type Number is use
d
--#   --internally and exclusively by the RSL to check if the current stack
--#   --has enough space:

--#   CHECK_STACK_INTERRUPT                    : constant := 96;

--#   --The software interrupt having the following Interrupt Type Number is use
d
--#   --internally and exclusively by the RSL to effect switching between tasks:

--#   DISPATCH_CODE_INTERRUPT                  : constant := 99;


--#   --Interrupts having the following Interrupt Type Numbers (all
--#   --software-generated) are used internally and exclusively by the generated
--#   --code for effecting subprogram entry sequences where there is no SFDD:

--#   ENTER_SUBPROGRAM_WITHOUT_LPP_INTERRUPT : constant := 97;
--# --The generated code uses this interrupt to effect a subprogram entry
--# --sequence without a Lexical Parent Pointer.

--#   ENTER_SUBPROGRAM_INTERRUPT               : constant := 98;
--# --The generated code uses this interrupt to effect a subprogram entry
--# --sequence with a Lexical Parent Pointer.


--#   --Interrupts having the following Interrupt Type Numbers (all software-
--#   --generated) are used internally and exclusively by the generated code to
--#   --cause certain Ada exceptions to be forced:

--#   PROGRAM_ERROR_INTERRUPT                  : constant := 102;
--# --This interrupt must be interpreted as the exception PROGRAM_ERROR.

--#   CONSTRAINT_ERROR_INTERRUPT               : constant := 103;
--# --This interrupt must be interpreted as the exception CONSTRAINT_ERROR.

--#   NUMERIC_ERROR_INTERRUPT                  : constant := 104;
--# --This interrupt must be interpreted as the exception NUMERIC_ERROR.


 -#   --Interrupts having the following Interrupt Type Numbers (all software-
 -#   --generated) are used internally and exclusively by the generated code to
 -#   --cause certain RSL services to be invoked:

 -#   ALLOCATE_OBJECT_INTERRUPT                : constant := 105;
 # --This interrupt causes an object to be allocated in the heap of the
 # --anonymous task.


      --The software interrupts having the following Interrupt Type Numbers are
 d
      --internally and exclusively by the RSL to effect entry to and exit from
      --Innocuous Critical Regions:

      ENTER_INNOCUOUS_CRITICAL_REGION_INTERRUPT: constant := 106;
```

```
--#   LEAVE_INNOCUOUS_CRITICAL_REGION_INTERRUPT: constant := 107;

--#   HALT_INTERRUPT                           : constant := 109;
--#   END_OF_PROGRAM_INTERRUPT                 : constant := 110;

--#STOP  PC_DOS


  --Intel "reserves" interrupts with Interrupt Type Numbers in the range 0..31,
  --with 32..255 available to the user.  We allow the user to equate interrupts
  --in the range 72..103 to entries of task via Ada address clauses.  We also
  --allow such use of interrupts 1, 2, and 3, as well as interrupts arriving at

  --#START iAPX86, iAPX286R, iAPX386R, iAPX286P, iAPX386P
      --PIC inputs IR1, IR2, IR3, IR4, IR5, and IR6 (Interrupt Type Numbers 65..
70).
  --#STOP  iAPX86, iAPX286R, iAPX386R, iAPX286P, iAPX386P

  --#START iAPX186
--#   --iAPX186 inputs INT0, INT1, and INT2 (Interrupt Type Numbers 12..14).
  --#STOP  iAPX186

  --#START PC_DOS
--#   -- IBM-PC DOS reserves interrupts with Interrupt Type Numbers in the
--#   -- range 0..95.  We allow the use of 1, 3, 6, 7, as well as
--#   -- interrupts arriving at PIC inputs IR2, IR3, IR4, IR5 (Interrupt
--#   -- Type Numbers 10, 11, 12, and 13).
  --#STOP  PC_DOS

pragma PAGE;


  --The enumeration literals of type ENTRY_KIND distinguish between entries of
  --software tasks and interrupt entries, and identify different varieties of
  --the latter when used as the second argument in a FAST_INTERRUPT_ENTRY
  --pragma:

  type ENTRY_KIND is
    (
      ----------------------------
      --ORDINARY INTERRUPT ENTRY--
      ----------------------------

        ORDINARY_INTERRUPT_ENTRY,
          --This is not a Fast Interrupt Entry.  It is invoked by an interrupt
          --other than NMI.  This entry may be called by a software task as
          --well as by interrupt.

          --If an interrupt is equated to an entry by means of an address
          --clause, and the FAST_INTERRUPT_ENTRY pragma is not given for that
          --entry, the entry will be treated as an ORDINARY_INTERRUPT_ENTRY by
          --default.

          --When this kind of interrupt entry occurs, the state of the 8087
          --Numeric Data Processor will always be saved as part of the context
          --of the interrupted task, because the normal task-switching
          --mechanism will attempt to restore it before resuming the
          --interrupted task.
          --This is a Non-Maskable Interrupt Entry invoked only by NMI whose
          --accept body makes no entry calls.

        NO_NDP_NON_MASKABLE
          --This is a Non-Maskable Interrupt Entry invoked only by NMI whose
          --accept body makes no entry calls.

          --It differs from NON_MASKABLE only in that the state of the 8087
          --Numeric_Data Processor is neither saved nor restored during
          --interrupt delivery.


    );

pragma PAGE;

-------------------------------------------------------------------------------
-- NOTE: Be sure to compute TICK and TICKS_PER_DAY by hand, as the roundoff   --
-- errors introduced in computer arithmetic are unacceptably inaccurate.      --
-------------------------------------------------------------------------------


  --#START iAPX86
      --If one loaded the Programmable Interval Timer (PIT) clock counter with t
he
      --shortest possible delay, namely 1, TICK is the amount of time, in second
s,
      --which would pass between the loading and the interrupt which the PIT wou
ld
      --issue upon counting down and reaching zero.

      TICK        : constant :=  6.510_416_666_666_666_666_667E-6;
    --roughly 6.5 microseconds
  --#STOP  iAPX86

  --#START iAPX186
--#   --For the system clock counter of the iAPX186's Internal Timer Unit, TICK
is
--#   --the amount of time, in seconds, that it takes to count from 0 to 1.

--#   --IMPORTANT: The iSBC 186/03A runs at 8 MHz, and its Internal Timer Unit's
--#   --base clock rate is 8 MHz divided by four, or 2 MHz.
--#   --Therefore one counter tick = 1 sec. / 2_000_000 = 0.000_000_5 sec.
--#   --One major clock cycle = 2**16  * one counter tick
--#   --                      = 65_536 * 0.000_000_5 sec.
--#   --                      = 0.032_768 sec.
--#   --We would like a greater time interval between counter interrupts used fo
r
--#   --timekeeping.  In fact, we would like about one second, or as close as
--#   --possible.  This means that we must prescale our system clock counter.
--#   --
--#   --To find prescale factor, solve for X:
--#   --    X * one major clock cycle = 1 second
--#   --    X * 0.032_768 sec.        = 1 sec.
--#   --    X                         = 1 / 0.032_768
--#   --    X                         = 30.517_578_125
--#   --    X                        ~= 30
--#   --
--#   --So SYSTEM.TICK = a prescaled counter tick
--#   --              = 30 * 0.000_000_5 sec.
--#   --              = 0.000_015 sec.
--#   --and a prescaled major clock cycle = 2**16  * one prescaled counter tick
--#   --                                  = 65_536 * 0.000_015 sec.
--#   --                                  ~= 0.983 sec.
--#   --
--#   --There are 66_666 + 2/3 ticks in a second.
--#   --The number of ticks per second must be used to calculate the values of t
he
--#   --ADA_RSL constants CLOCK_TICKS_PER_DAY, TICKS_PER_HALF_DAY, and INT_CHUNK_
--#   --RAW_TIME.

--#   TICK        : constant := 0.000_015;  --15 microseconds
  --#STOP  iAPX186

  --#START iAPX286P, iAPX386P, iAPX286R, iAPX386R
--#   --If one loaded the Programmable Interval Timer (PIT) clock counter with t
he
--#   --shortest possible delay, namely 1, TICK is the amount of time, in second
s,
--#   --which would pass between the loading and the interrupt which the PIT wou
ld
--#   --issue upon counting down and reaching zero.

--#   --The CLK0 input to the 8254 PIT on the iSBC 286/10 is 1.23 MHz.
--#   --So one counter 0 tick = 1 sec. / 1_230_000 = 0.0000_00813_00813_... sec.
--#   --One major clock cycle = 2**16  * one counter tick
--#   --                      = 65_536 * 0.0000_00813_00813_... sec.
--#   --                      ~= 0.0535 sec.
--#   --
--#   --There are 1_230_000 (in hex, 16#0012_C4B0#) ticks in a second if
--#   --is not prescaled.

--#   --The maximum recommended value of the smallest delay duration (LRM 9.6) i
s
--#   --50 microseconds.  This will give the lowest possible frequency of timer
--#   --interrupts.  To achieve this, another counter is needed as a prescaler.
 The
--#   --prescale factor (X) is calculated as follows.
--#   --   X = 0.0000_5 / One counter 0 tick
--#   --   X = 0.0000_5 / 0.0000_00813_00813_....
--#   --   X = 61.5
--#   --   X = 61  (nearest rounded off value)
--#   --Therefore SYSTEM.TICK = 61 * counter 0 tick
--#   --                      = 61 * 0.0000_00813_00813_... sec.
--#   --                      = 0.0000_49593_49593_49593_... sec.
--#   --                      = 49.593_49593_49593_49593_...  microseconds
--#   --One major clock cycle = 2**16  * SYSTEM.TICK
--#   --                      = 65_536 * 0.0000_49593_49593_49593_... second
--#   --                      = 3.2501_59349_59349_59349_... seconds
    --#   TICK        : constant := 0.0000_49593_49593_49593;  --about 49.59 mi
croseconds

--#   TICKS_PER_SECOND : constant := 20163.93442_62209_52836_06557;  --approxima
te

--#   --TICKS_PER_SECOND must be used to calculate (by hand!) the values of the
--#   --ADA_RSL constants CLOCK_TICKS_PER_DAY, TICKS_PER_HALF_DAY, and INT_CHUNK_
--#   --RAW_TIME.

  --#STOP  iAPX286P, iAPX386P, iAPX286R, iAPX386R
  --#START PC_DOS
--#   --If one loaded the Programmable Interval Timer (PIT) clock counter with t
he
--#   --shortest possible delay, namely 1, TICK is the amount of time, in second
s,
--#   --which would pass between the loading and the interrupt which the PIT wou
ld
--#   --issue upon counting down and reaching zero.  The clock input to the
--#   --PIT is 1.19318 MHZ, so a tick is  1/1.19318 MHZ or approximately
--#   --0.8380965E-6 seconds

--#   TICK        : constant :=  0.838096515E-6;
--# --roughly .83 microseconds
  --#STOP  PC_DOS



  type TIME is private;
  NULL_TIME : constant TIME;

  type DIRECTION_TYPE is( AUTO_INCREMENT, AUTO_DECREMENT );
  type PARITY_TYPE    is( ODD, EVEN );

  type FLAGS_REGISTER is
     record

        --#START iAPX286P, iAPX386P
--#     NESTED_TASK         : BOOLEAN        := FALSE;
--#     IO_PRIVILEGE_LEVEL : NATURAL range 0..3 := 1;
        --#STOP  iAPX286P, iAPX386P

        OVERFLOW  : BOOLEAN                  := FALSE;
        DIRECTION : SYSTEM.DIRECTION_TYPE := SYSTEM.AUTO_INCREMENT;
        INTERRUPT : BOOLEAN                  := TRUE;
        TRAP      : BOOLEAN                  := FALSE;
        SIGN      : BOOLEAN                  := FALSE;
        ZERO      : BOOLEAN                  := TRUE;  --nihilistic view
        AUXILIARY : BOOLEAN                  := FALSE;
        PARITY    : SYSTEM.PARITY_TYPE       := SYSTEM.EVEN;
        CARRY     : BOOLEAN                  := FALSE;
     end record;

  for  FLAGS_REGISTER use
     record

        --#START iAPX286P, iAPX386P
--#     NESTED_TASK        at 0 range 14..14;
--#     IO_PRIVILEGE_LEVEL at 0 range 12..13;
        --#STOP  iAPX286P, iAPX386P

        OVERFLOW           at 0 range 11..11;
        DIRECTION          at 0 range 10..10;
        INTERRUPT          at 0 range  9.. 9;
        TRAP               at 0 range  8.. 8;
        SIGN               at 0 range  7.. 7;
        ZERO               at 0 range  6.. 6;
        AUXILIARY          at 0 range  4.. 4;
        PARITY             at 0 range  2.. 2;
        CARRY              at 0 range  0.. 0;
```

```
      end record;

    NORMALIZED_FLAGS_REGISTER : constant SYSTEM.FLAGS_REGISTER :=
      (
        --#START iAPX286P, iAPX386P
--#     NESTED_TASK         => FALSE,
--#     IO_PRIVILEGE_LEVEL => 1,
        --#STOP  iAPX286P, iAPX386P

        OVERFLOW  => FALSE,
        DIRECTION => SYSTEM.AUTO_INCREMENT,
        INTERRUPT => TRUE,
        TRAP      => FALSE,
        SIGN      => FALSE,
        ZERO      => TRUE,  --nihilistic view
        AUXILIARY => FALSE,
        PARITY    => SYSTEM.EVEN,
        CARRY     => FALSE
      );


    subtype PRIORITY is INTEGER range 1..15;

    UNRESOLVED_REFERENCE: exception;                  --see Appendix 30 of A-spec
    SYSTEM_ERROR        : exception;



  function EFFECTIVE_ADDRESS
    ( A: in SYSTEM.ADDRESS
    )
  return SYSTEM.ABSOLUTE_ADDRESS;

    --PURPOSE:
    --  This function, written in ASM86, returns the 20-bit effective address
    --  specified by the segment/offset register pair A.
  pragma INTERFACE( ASM86, EFFECTIVE_ADDRESS );


  function FAST_EFFECTIVE_ADDRESS
  -- ( A: in SYSTEM.ADDRESS
         --found in DX (segment part) and AX (offset part), NOT on stack
  -- )
  return SYSTEM.ABSOLUTE_ADDRESS;
     --in DX:AX;

    --PURPOSE:
    --  This function, written in ASM86, returns the 20-bit effective address
    --  specified by the segment/offset register pair DX:AX.
    --  This function is intended for use by ASM routines.  It does not observe
    --  Ada calling conventions and therefore does not make a null SFDD.  It
    --  does save and later restore all those registers that it uses
    --  internally.
  pragma INTERFACE( ASM86, FAST_EFFECTIVE_ADDRESS );


  function TWOS_COMPLEMENT_OF
    ( W: in SYSTEM.WORD
    )
  return SYSTEM.WORD;
  --PURPOSE:
  --  This function, written in ASM86, returns the two's complement of the
  --  given argument.
  --ASSUMPTIONS:
  --  1) CRITICAL REGION INFORMATION:
  --     This procedure makes no assumptions about critical regions.
  --     It neither enters nor leaves a critical region.
pragma INTERFACE( ASM86, TWOS_COMPLEMENT_OF );


procedure ADD_TO_ADDRESS
  ( ADDR  : in out SYSTEM.ADDRESS;
    OFFSET: in     SYSTEM.OFFSET_REGISTER );

  --PURPOSE:
  --  This procedure, written in ASM86, adds OFFSET to the offset part of
  --  ADDR.  If overflow occurs, NUMERIC_ERROR is raised.
  --SIDE EFFECTS:
  --  Raising of NUMERIC_ERROR.
pragma INTERFACE( ASM86, ADD_TO_ADDRESS );


procedure SUBTRACT_FROM_ADDRESS
  ( ADDR  : in out SYSTEM.ADDRESS;
    OFFSET: in     SYSTEM.OFFSET_REGISTER );

  --PURPOSE:
  --  This procedure, written in ASM86, subtracts OFFSET from the offset part
  --  of ADDR.  If underflow occurs, NUMERIC_ERROR is raised.
  --SIDE EFFECTS:
  --  Raising of NUMERIC_ERROP.
pragma INTERFACE( ASM86, SUBTRACT_FROM_ADDRESS );


function INTERRUPT_TYPE_NUMBER_OF
  ( A : in SYSTEM.ADDRESS
  )
return SYSTEM.INTERRUPT_TYPE_NUMBER;

  --PURPOSE:
  --  This function, written in ASM86, returns the Interrupt Type Number that
  --  uniquely identifies the interrupt whose interrupt vector is located at
  --  the specified address.  If this address is not the address of an
  --  interrupt vector, CONSTRAINT_ERROR is raised.
  --SIDE EFFECTS:
  --  Raising of CONSTRAINT_ERROR.
pragma INTERFACE( ASM86, INTERRUPT_TYPE_NUMBER_OF );


procedure GET_ADDRESS_FROM_INTERRUPT_TYPE_NUMBER
  ( A   : out SYSTEM.ADDRESS;
    ITN : in  SYSTEM.INTERRUPT_TYPE_NUMBER
  );

  --PURPOSE:
  --  This procedure, written in ASM86, returns the address of the interrupt
  --  vector numbered ITN.
pragma INTERFACE( ASM86, GET_ADDRESS_FROM_INTERRUPT_TYPE_NUMBER );
```

```
function GREATER_THAN
  ( A1 : in SYSTEM.ADDRESS;
    A2 : in SYSTEM.ADDRESS
  )
return BOOLEAN;

  --PURPOSE:
  --  This function, written in ASM86, returns the value of the expression
  --  A1 > A2;
pragma INTERFACE( ASM86, GREATER_THAN );


function MINUS
  ( A1 : in SYSTEM.ADDRESS;
    A2 : in SYSTEM.ADDRESS
  )
return LONG_INTEGER;

  --PURPOSE:
  --  This function, written in ASM86, returns the signed value of A1 - A2.
pragma INTERFACE( ASM86, MINUS );


function ">"
  ( A1 : in SYSTEM.ADDRESS;
    A2 : in SYSTEM.ADDRESS
  )
return BOOLEAN renames SYSTEM.GREATER_THAN;


function "-"
  ( A1 : in SYSTEM.ADDRESS;
    A2 : in SYSTEM.ADDRESS
  )
return LONG_INTEGER renames SYSTEM.MINUS;



--   procedure ADJUST_FOR_UPWARD_GROWTH
--     ( OLD_ADDRESS      : in  SYSTEM.ADDRESS;
--       ADJUSTED_ADDRESS: out SYSTEM.ADDRESS );
    --  Transforms the given SYSTEM.ADDRESS into a representation yielding
    --  the same effective address, but in which the SEGMENT component is
    --  as large as possible.

--   procedure ADJUST_FOR_DOWNWARD_GROWTH
--     ( OLD_ADDRESS      : in  SYSTEM.ADDRESS;
--       ADJUSTED_ADDRESS: out SYSTEM.ADDRESS );
    --  Transforms the given SYSTEM.ADDRESS into a representation yielding
    --  the same effective address, but in which the OFFSET component is as
    --  large as possible.

--private

-- pragma INTERFACE( ASM86, ADJUST_FOR_UPWARD_GROWTH );
-- pragma INTERFACE( ASM86, ADJUST_FOR_DOWNWARD_GROWTH );

private
```

```
    type LONG_CYCLE is array(1..3)of SYSTEM.WORD;
    pragma PACK( LONG_CYCLE );  --Make this type occupy 64 bits.

    type TIME is  --This may be viewed as a single 64-bit integer
      record          --representing a quantity of SYSTEM.TICKs.
        CYCLES : LONG_CYCLE;
        TICKS  : SYSTEM.WORD;
      end record;

    for  TIME use record
        CYCLES at 0 range 0..47;
        TICKS  at 6 range 0..15;
      end record;


    --A TIME variable may be viewed as a 64-bit integer, or as a record with a
    --more significant CYCLES part and a less significant TICKS part.  Whenever
    --the TICKS part is incremented, the addition may carry over into the
    --adjacent CYCLEs part.
    --
    --Storage layout of a variable of type TIME:
    --
    --
    --                             increasing addresses
    --                            ----------------------->
    --
    --     +--------------+--------------+--------------+--------------+
    --     | CYCLES(1)    | CYCLES(2)    | CYCLES(3)    |   TICKS      |
    --     +--------------+--------------+--------------+--------------+
    --
    --      \_____   _____/
    --            \/
    --         one word

    NULL_TIME : constant TIME := ( (OTHERS => 0), 0 );

 end SYSTEM;
```

vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
(4) Representation Clause Restrictions
^^

++
Representation clauses specify how the types of the language are to be mapped onto the underlying machine. The following are restrictions on representation clauses.
++

Address Clauses

Address clauses are supported for the following items:

1. Scalar or composite objects with the following restrictions:

   (a) The object must not be nested within a subprogram or task directly or indirectly.

   (b) The size of the object must be determinable at time of compilation.

2. Subprograms with the following restrictions:

   (a) The subprogram can not be a library subprogram (LRM requirement).

   (b) Any subprogram declared within a subprogram having an address clause will be placed in relocatable sections.

3. Entries - An address clause may specify a hardware interrupt with which the entry is to be associated.

Length Clause

T'STORAGE_SIZE for task type T specifies the number of bytes to be allocated for the run-time stack of each task object of type T.

Enumeration Representation Clause

In the absence of a representation specification for an enumeration type T, the internal representation of T'FIRST is 0. The default SIZE for a stand-alone object of enumeration type T will be the smallest of the values 8, 16, or 32, such that the internal representation of T'FIRST and T'LAST both fall within the range:

```
    -2**(T'SIZE - 1) .. 2**(T'SIZE - 1)-1.
```

Length specifications of the form:

```
    for T'SIZE use N;
```

and/or enumeration representations of the form:

```
    for T use aggregate
```

Are permitted for N in 2..32, provided the representations and the SIZE conform to the relationship specified above, or else for N in 1..31, provided that the internal representation of T'FIRST > = 0 and the representation of T'LAST = 2**(T'SIZE) - 1.

For components of enumeration types within packed composite objects, the smaller of the default stand-alone SIZE and the SIZE from a length specification is used.

In accordance with the rules of Ada, and the implementation of package STANDARD, enumeration representation on types derived from the predefined type BOOLEAN are not accepted, but length specifications are accepted.

Record Representation Clause

A length specification of the form

```
for T'SIZE use N;
```

Will cause arrays and records to be packed, if required, to accommodate the length specification.

The PACK pragma may be used to minimize wasted space between components of arrays and records. The pragma causes the type representation to be chosen such that storage space requirements are minimized at the possible expense of data access time and and code space.

A record type representation specification may be used to describe the allocation of components in a record. Bits are numbered 0..7 from the right. (Bit 8 starts at the right of the next higher-numbered byte.)

The alignment clause of the form:

```
at mod N
```

can specify alignment of 1 (byte) or 2 (word).

```
vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
(5)   Conventions
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
The following conventions are used for an implementation-
generated name denoting implementation-dependent components.
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

         NONE

vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
(6)   Address Clauses
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
The following are conventions that define the interpretation
of expressions that appear in address clauses, including
those for interrupts.
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

         NONE

vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
(7)   Unchecked Conversions
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
The following are restrictions on unchecked conversion,
including those depending on the respective sizes of objects
of the source and target.
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

         A program is erroneous if it performs UNCHECKED-CONVERSION when
         the size of the source and target types have different.

vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
(8)   Input-Output Packages
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
The following are implementation-dependent characteristics
of the input-output packages.
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

         SEQUENTIAL_IO_Package

                               NOT SUPPORTED

+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Declare file type and applicable operations for files of
this type.
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

         DIRECT_IO Package

                               NOT SUPPORTED
```

TEXT_IO Package

```
--* PACKAGE SPECIFICATION FOR TEXT_IO

******************************************************************************

      The Specification of the Package TEXT_IO contains the following
      (implementation specific) definitions in addition to those specified
      in 14.3.10 of the LRM:

******************************************************************************

--#  Copyright 1986 Softech,  Inc., all rights reserved.

-- Copyright (C) 1987, SofTech, Inc.

with ADA_RSL, IO_EXCEPTIONS;

--#START iAPX86, iAPX186, iAPX286R, iAPX386R, iAPX286P, iAPX386P
    with SYSTEM, IO_DEFS;
--#STOP  iAPX86, iAPX186, iAPX286R, iAPX386R, iAPX286P, iAPX386P

--#START PC_DOS
--# with SYSTEM, IO_DEFS, BASIC_IO;
--#STOP  PC_DOS


--* PACKAGE SPECIFICATION FOR TEXT_IO

--* PURPOSE:
--%    This package provides input and output services for textual files
--%
--%    including creation,deletion,opening, and closing of said files.
--%    This package is as specified in the Ada Reference Manual (1982).
--%
--%    And here a word about primary and secondary routines.  A primary routine
is
--%    always visible outside the package.  If it references a file, it will
--%    attempt to gain exclusive access to that file descriptor.  (The term
--%    "exclusive access" is used with regard to tasks.)  All modifications or
--%    tests on file descriptor FIELDs must be made only if the current task
--%    has exclusive access to that descriptor.  In every case where a primary
--%    routine gains exclusive access to a file descriptor, that routine must
--%    release the file descriptor beFORE exiting.  Primary routines may call
--%    primary or secondary routines.  Secondary routines are never visible
--%    outside the package.  If a secondary routine references a file descriptor
,
--%    that routine assumes exclusive access for that descriptor.  Secondary
--%    routines may only call other secondary routines.  All calls to BASIC_IO
--%    for reading or writing are made by secondary routines.  All other
--%    BASIC_IO calls are made by primary routines.
--
```

```
PRAGMA PAGE;

--* SPECIFICATION:

PACKAGE text_io IS
USE  ada_rsl,

   TYPE file_type IS LIMITED PRIVATE;

   TYPE file_mode IS (in_file, out_file);

   TYPE count IS RANGE 0 .. integer'LAST;

   SUBTYPE positive_count IS count RANGE 1 .. count'LAST;

   unbounded : CONSTANT count := 0;      -- line and page length

   SUBTYPE field IS integer RANGE 0 .. integer'LAST;

   SUBTYPE number_base IS integer RANGE 2 .. 16;

   TYPE type_set IS (lower_case,upper_case);

   -- File Management -----------------------------------------------------

   PROCEDURE create ( file : IN OUT file_type;
                      mode : IN file_mode := out_file;
                      name : IN string := "";
                      form : IN string := "" );

   PROCEDURE open   ( file : IN OUT file_type;
                      mode : IN file_mode;
                      name : IN string;
                      form : IN string := "" );

   PROCEDURE close  ( file : IN OUT file_type );

   PROCEDURE delete ( file : IN OUT file_type );

   PROCEDURE reset  ( file : IN OUT file_type;
                      mode : IN file_mode );

   PROCEDURE reset  ( file : IN OUT file_type );

   FUNCTION mode    ( file : IN file_type ) RETURN file_mode;

   FUNCTION name    ( file : IN file_type ) RETURN string;

   FUNCTION form    ( file : IN file_type ) RETURN string;

   FUNCTION is_open ( file : IN file_type ) RETURN boolean;

   -- Control of default input and output files ---------------------------

   PROCEDURE    set_input       ( file : IN file_type );
   PROCEDURE    set_output      ( file : IN file_type );

   FUNCTION     standard_input  RETURN file_type;
FUNCTION     standard_output RETURN file_type;

FUNCTION     current_input   RETURN file_type;
FUNCTION     current_output  RETURN file_type;

-- Specification of line and page lengths ------------------------------

PROCEDURE    set_line_length ( file : IN file_type;
                               to   : IN count );

PROCEDURE    set_line_length ( to   : IN count );  -- for default output file

PROCEDURE    set_page_length ( file : IN file_type;
                               to   : IN count );

PROCEDURE    set_page_length ( to   : IN count );   -- for default output fil
e

FUNCTION     line_length     ( file : IN file_type ) RETURN count;

FUNCTION     line_length     RETURN count;         -- for default output file

FUNCTION     page_length     ( file : IN file_type ) RETURN count;

FUNCTION     page_length     RETURN count;


-- Column, Line, and Page Control ----------------------------------------

PROCEDURE    new_line        ( file    : IN file_type;
                               spacing : IN positive_count := 1 );

PROCEDURE    new_line        ( spacing : IN positive_count := 1 );

PROCEDURE    skip_line       ( file    : IN file_type;
                               spacing : IN positive_count := 1 );

PROCEDURE    skip_line       ( spacing : IN positive_count := 1 );

FUNCTION     end_of_line     ( file : IN file_type) RETURN BOOLEAN;

FUNCTION     end_of_line     RETURN boolean;

PROCEDURE    new_page        ( file : IN file_type );

PROCEDURE    new_page;                             -- default output file

PROCEDURE    skip_page       ( file : IN file_type );

PROCEDURE    skip_page;                            -- default input file

FUNCTION     end_of_page     ( file : IN file_type ) RETURN boolean;

FUNCTION     end_of_page     RETURN boolean;       -- default input file

FUNCTION     end_of_file     ( file : IN file_type ) RETURN boolean;

FUNCTION     end_of_file     RETURN boolean;       -- default input file

PROCEDURE    set_col         ( file : IN file_type;
                                      to    : IN positive_count );

   PROCEDURE    set_col          ( to    : IN positive_count );  -- for default ou
tput file

   PROCEDURE    set_line         ( file : IN file_type;
                                   to    : IN positive_count );

   PROCEDURE    set_line         ( to    : IN positive_count );  -- for default ou
tput file

   FUNCTION     col              ( file : IN file_type ) RETURN positive_count;

   FUNCTION     col              RETURN positive_count;          -- for default ou
tput file

   FUNCTION     line             ( file : IN file_type ) RETURN positive_count;

   FUNCTION     line             RETURN positive_count;          -- for default ou
tput file

   FUNCTION     page             ( file : IN file_type ) RETURN positive_count;

   FUNCTION     page             RETURN  positive_count;         -- default output
 file

   -- CHARACTER input_output -----------------------------------------------

   PROCEDURE    get       ( file   : IN file_type;
                            item   : OUT character );

   PROCEDURE    get       ( item   : OUT character );

   PROCEDURE    put       ( file   : IN file_type;
                            item   : IN character );

   PROCEDURE    put       ( item   : IN character );

   -- STRING input_output --------------------------------------------------

   PROCEDURE    get       ( file   : IN file_type;
                            item   : OUT string );

   PROCEDURE    get       ( item   : OUT string );

   PROCEDURE    put       ( file   : IN file_type;
                            item   : IN string );

   PROCEDURE    put       ( item   : IN string );

   PROCEDURE    get_line ( file   : IN  file_type;
                           item   : OUT string;
                           last   : OUT natural );

   PROCEDURE    get_line ( item   : OUT string;
                           last   : OUT natural );

   PROCEDURE    put_line ( file   : IN  file_type;
                           item   : IN  string );
```

```
PROCEDURE    put_line ( item   : IN  string );

--------------------------------------------------------------------------------

-- Generic package for Input_out of Integer Types

GENERIC
   TYPE num IS RANGE <>;

PACKAGE integer_io IS                          -- I N T E G E R _ I O
   default_width : field := num'WIDTH;
   default_base  : number_base := 10;

   PROCEDURE  get ( file     : IN file_type;
                    item     : OUT num;
                    width    : IN field := 0 );

   PROCEDURE  get ( item     : OUT num;
                    width    : IN field := 0 );

   PROCEDURE  put ( file     : IN file_type;
                    item     : IN num;
                    width    : IN field := default_width;
                    base     : IN number_base := default_base );

   PROCEDURE  put ( item     : IN num;
                    width    : IN field := default_width;
                    base     : IN number_base := default_base );

   PROCEDURE  get ( from     : IN string;
                    item     : OUT num;
                    last     : OUT positive );

   PROCEDURE  put ( to       : OUT string;
                    item     : IN num;
                    base     : IN number_base := default_base );

END integer_io;

--------------------------------------------------------------------

-- Generic packages for Input_ouput of Real Type

GENERIC
   TYPE num IS DIGITS <>;

PACKAGE float_io IS
   default_fore  : field := 2;
   default_aft   : field := num'DIGITS - 1;
   default_exp   : field := 3;

   PROCEDURE get ( file     : in file_type;
                   item     : OUT num;
                   width    : IN field := 0 );

   PROCEDURE get ( item     : OUT num;
                   width    : IN field := 0 );

   PROCEDURE put ( file     : IN file_type;
                   item     : IN num;
```

```
                       fore      : IN field := default_fore;
                       aft       : IN field := default_aft;
                       exp       : IN field := default_exp );

   PROCEDURE put ( item      : IN num;
                   fore      : IN field := default_fore;
                   aft       : IN field := default_aft;
                   exp       : IN field := default_exp );

   PROCEDURE get ( from      : IN string;
                   item      : OUT num;
                   last      : OUT positive );

   PROCEDURE put ( TO        : OUT string;
                   item      : IN num;
                   aft       : IN field := default_aft;
                   exp       . IN field := default_exp );

END float_io;

-----------------------------------------------------------------

GENERIC
   TYPE num IS DELTA <>;

PACKAGE fixed_io IS
   default_fore  : field := num'FORE;
   default_aft   : field := num'AFT;
   default_exp   : field := 0;

   PROCEDURE get ( file      : IN file_type;
                   item      : OUT num;
                   width     · IN field := 0 );

   PROCEDURE get ( item      : OUT num;
                   width     : IN field := 0 );

   PROCEDURE put ( file      : IN file_type;
                   item      : IN num;
                   fore      : IN field := default_fore;
                   aft       : IN field := default_aft;
                   exp       : IN field := default_exp );

   PROCEDURE put ( item      : IN num;
                   fore      : IN field := default_fore;
                   aft       : IN field := default_aft;
                   exp       : IN field := default_exp );

   PROCEDURE get ( from      : IN string;
                   item      : OUT num;
                   last      : OUT positive );

   PROCEDURE put ( to        : OUT string;
                   item      : IN num;
                   aft       : IN field := default_aft;
                   exp       : IN field := default_exp );

END fixed_io;

-----------------------------------------------------------------
```

```
-- Generic package for Input_Output of Enumeration Types

GENERIC
   TYPE enum IS (<>);

PACKAGE enumeration_io IS
   default_width      : field    := 0;
   default_setting    : type_set := upper_case;

   PROCEDURE get ( file     : IN  file_type;
                   item     : OUT enum );

   PROCEDURE get ( item     : OUT enum );

   PRCCEDURE put ( file     : IN file_type;
                   item     : IN enum;
                   width    : IN field    := default_width;
                   set      : IN type_set := default_setting );

   PROCEDURE put ( item     : IN enum;
                   width    : IN field    := default_width;
                   set      : IN type_set := default_setting );

   PROCEDURE get ( from     : IN string;
                   item     : OUT enum;
                   last     : OUT positive );

   PROCEDURE put ( to       : OUT string;
                   item     : IN enum;
                   set      : IN type_set := default_setting );

END enumeration_io;

--------------------------------------------------------------------------------

-- Exceptions

   status_error : EXCEPTION RENAMES io_exceptions.status_error;
   mode_error   : EXCEPTION RENAMES io_exceptions.mode_error;
   name_error   : EXCEPTION RENAMES io_exceptions.name_error;
   use_error    : EXCEPTION RENAMES io_exceptions.use_error;
   device_error : EXCEPTION RENAMES ic_exceptions.device_error;
   end_error    : EXCEPTION RENAMES io_exceptions.end_error;
   data_error   : EXCEPTION RENAMES io_exceptions.data_error;
   layout_error : EXCEPTION RENAMES io_exceptions.layout_error;

--
```

```
----------------------P R I V A T E -----------------------------------

PRIVATE

-- REPRESENTATION OF TEXT_IO FILES:
-- -------------------------------

-- This implementation of TEXT_IO is for the Intel targets.  For
-- input files, a variety of possible file formats are supported.
-- For output, a single canonical format corresponding to the format
-- of DOS produced text files is used.
--
--
--                TEXT_IO OUTPUT FILE FORMAT
--                -------------------------
--
--
--          file      ::=  page {eop page} eof
--
--          page      ::=  line {eol line}
--
--          line      ::=  {character}
--
--          eol       ::=  ASCII.CR  ASCII.LF
--
--          eop       ::=  ASCII.FF
--
--          eof       ::=  ASCII.SUB
--
--          character ::=  any ASCII character except CR, LF, FF, and SUB
--
--     Note that for an output file, a physical line terminator ends
--     every line except the last line in each page.  A physical page
--     terminator follows every page except the last page which is
--     terminated by the physical file terminator.  The final page
--     terminator is omitted in keeping with common practice.
--
--     An empty physical file logically consists of an Ada line terminator
--     followed by a page terminator, followed by a file terminator.
--     A physical file containing only a form feed character logically consists
--     of two pages, each containing a single line empty line.
--
--
--               TEXT_IO INPUT FILE FORMATS
--               -------------------------
--
--   The PHYSICAL syntax for an INPUT file is broad enough to accept a variety
--   of possible text file forms including some which are not produced by
--   TEXT_IO.  The following physical text patterns are interpreted as Ada
--   logical lines, pages and files by TEXT_IO when reading files:
--
--          file      ::=   page {eop page} eof
--
--          page      ::=   line {eol line}
--
--          line      ::=   {character}
--
--          eol       ::=   ASCII.CR ASCII.LF
--                       | ASCII.CR
--                       | ASCII.LF
--
--          eop      ::=   ASCII.FF
--
--          eof      ::=   ASCII.SUB
--                       | (end of data condition)
--
--          character ::= any character except ASCII: CR, LF, FF, SUB.
--
--
--  Thus for an input file, a line may be explicity terminated by a carriage
--  return/line feed pair, by carriage return alone, or by line feed alone.
--  An end of line is always implicit in a form feed or the physical end of
--  file.
--
--  A file may be explicitly terminated by a control Z character or
--  implicity when the end of input data is encountered.  However, an
--  embedded control Z character will be treated as the end of file even
--  though it may not be the physical end of data.  The end of file is
--  always preceded by an implicit logical line terminator and page terminator.
--
--  The procedure READ_CHAR generates a page_term character corresponding
--  tothe implicit page terminator which precedes the end for file.
--  The implicit LINE_TERMINATOR which precedes each page terminator is
--  not generated READ_CHAR.
--
--  In the implementation of TEXT_IO, the code which interprets or
--  produces the physical file syntax has been isolated in the
--  following procedures:
--
--          read_char    - gets the next input character or teminator.
--          end_of_line  - checks if a line, page or file terminator is next.
--          end_of_page  - checks if a page or file terminator follows.
--          end_of_file  - checks if a file terminator follows.
--          txt_put_char - output a logical character.
--          txt_new_line - starts a new line.
--          txt_new_page - starts a new page.
--          write_char   - puts the next physical character.
--

--  Private Data:

   buffer_length : CONSTANT := 256;

   max_line_length : CONSTANT := buffer_length;

   TYPE char_buffer IS ARRAY (integer RANGE 1..buffer_length) OF character;

   TYPE file_rec IS -- common file state description; actual FILE_TYPE
      RECORD        -- declarations will be access types to this record.

         --#START PC_DOS
--#      stream          : basic_io.stream_type;
--#                                                -- BASIC_IO file handle.
         --#STOP  PC_DOS

         --#START iAPX86, iAPX186, iAPX286, iAPX286R, iAPX386R, iAPX286P, iAPX38
6P
         stream          : io_defs.stream_id_prv;
```

```
        --#STOP  iAPX86, iAPX186, iAPX286, iAPX286R, iAPX386R, iAPX286P, iAPX38
6P

        mode              : file_mode;       -- IN_FILE or OUT_FILE.

        curr_col          : count := 1;      -- Next column to be read
                                             -- or written.
        curr_line         : count := 1;      -- Current line in page.
        curr_page         : count := 1;      -- Current page in file.

        line_len          : count := unbounded;
                                             -- TEXT_IO line_length
        page_len          : count := unbounded;
                                             -- TEXT_IO page_length

        curr_rec_length : integer := 0;      -- Index of last character in
                                             -- in TEXT_BUF (when reading)
        text_index        : integer := 1;    -- Index of next character in
                                             -- TEXT_BUF to be read or
                                             -- written.
        text_buf          : char_buffer;     -- Input/outpt buffer.
        prev_char         : character := ASCII.NUL;
                                             -- Previous character returned
                                             -- by READ_CHAR.
        pending_terminator : character := ASCII.NUL;
                                             -- A terminator which has been
                                             -- passed to WRITE_CHAR but not
                                             -- yet placed in the text buffer.
                                             -- Value may be LINE_TERM,
                                             -- PAGE_TERM or ASCII.NUL
                                             -- indicating no pending
                                             -- terminator.
        back_up           : boolean := false;
                                             -- True if TXT_BACK_UP has been
                                             -- called to cause PREV_CHAR to
                                             -- be re-read.
        at_eof            : boolean := false;
                                             -- Set true when READ_CHAR sees
                                             -- the end of file marker.
     END RECORD;

TYPE file_type IS ACCESS file_rec;

---------------------------------------------------------------------------

std_input    : file_type;          -- the standard and current file descriptors
std_output   : file_type;          -- should not be visible to the user except
curr_input   : file_type;          -- through the provided procedure (see above).
curr_output  : file_type;

-- Define logical file marker values.

line_term    : CONSTANT character := ASCII.LF;
page_term    : CONSTANT character := ASCII.FF;    -- form feed     (ctrl-L) (16#0
C#)
file_term    : CONSTANT character := ASCII.SUB;   --               (ctrl-Z) (16#1
A#)

TYPE character_set IS ARRAY (character) OF BOOLEAN;
```

```
-- The TERMINATOR array is used to quickly determine whether a character is
-- is a physical terminator.

terminator   : CONSTANT character_set := character_set'
               (ASCII.CR |
                ASCII.LF |
                ASCII.FF |
                ASCII.SUB => TRUE,
                OTHERS    => FALSE);

-- The SPACE_ETC array is used to quickly determine whether a character is
-- to be skipped because its a space, tab, vertical tab, or terminator.

space_etc    : CONSTANT character_set := character_set'
               (' '      |
                ASCII.HT |
                ASCII.VT |
                ASCII.CR |
                ASCII.LF |
                ASCII.FF |
                ASCII.SUB => TRUE,
                OTHERS    => FALSE);
END text_io;
```

LOW_LEVEL_IO

++
Include either the LOW_LEVEL_IO package specification or the following sentence:

Low-level input-output is not provided.
++

```
--# Copyright 1986 Softech, Inc., all rights reserved.

-- Copyright (C) 1987, SofTech, Inc.


with SYSTEM; use SYSTEM;
--* PACKAGE SPECIFICATION FOR LOW_LEVEL_IO


--* PURPOSE:
--%   To support the programming of devices that can be accessed through ports
--%   in the memory space and the I/O space of the iAPX186. Specific devices
--%   or device types that cannot be assumed to be present in all iAPX186-based
--%   targets should be supported by specific packages (e.g., MPSC).
--
pragma PAGE;     -- In package LOW_LEVEL_IO

--* SPECIFICATION:

package LOW_LEVEL_IO is

   --Support for I/O-mapped input and output:
   procedure SEND_CONTROL    ( DEVICE : in IO_ADDRESS; DATA : in out BYTE );
   procedure SEND_CONTROL    ( DEVICE : in IO_ADDRESS; DATA : in out WORD );
   procedure RECEIVE_CONTROL( DEVICE : in IO_ADDRESS; DATA : in out BYTE );
   procedure RECEIVE_CONTROL( DEVICE : in IO_ADDRESS; DATA : in out WORD );

   --Support for memory-mapped input and output:
   procedure SEND_CONTROL    ( DEVICE : in ADDRESS; DATA : in out BYTE );
   procedure SEND_CONTROL    ( DEVICE : in ADDRESS; DATA : in out WORD );
   procedure RECEIVE_CONTROL( DEVICE : in ADDRESS; DATA : in out BYTE );
   procedure RECEIVE_CONTROL( DEVICE : in ADDRESS; DATA : in out WORD );

end LOW_LEVEL_IO;
```

```
vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
(9)   Package STANDARD
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

-- Copyright (C) 1986, SofTech, Inc.

PACKAGE standard IS

*********************************************************************************

      The Package STANDARD contains the following (implementation specific)
      definitions in addition to those specified in Annex C of the LRM:

*********************************************************************************

    TYPE integer IS RANGE -32_768 .. 32_767;


    FOR  integer'SIZE USE 16;


    TYPE long_integer IS RANGE -2_147_483_648 .. 2_147_483_647;


    TYPE float IS DIGITS 6 RANGE
             -(2#1.111_1111_1111_1111_1111_1111#E+127) ..
              (2#1.111_1111_1111_1111_1111_1111#E+127);

--  Type float is realized using the Intel machine type SHORT REAL.
--  SHORT REAL provides 24 bits of mantissa (one bit is  implied),
--  and it provides 8 bits for a biased exponent.  However only the values
--  1..254 are exponents of normalized numbers.  The bias is 127, so the
--  exponent range is -126..127.
--  This leads to the following attributes for the type float:
--              float'digits  = 6   [LRM 3.5.7, 3.5.8]
--            float'mantissa  = 21  [LRM 3.5.7, 3.5.8]
--                float'emax  = 84  [LRM 3.5.8]
--             float'epsilon  = 2.0 ** (-20) [LRM 3.5.8]
--                            = 2#1.000_0000_0000_0000_0000_0000#E-20
--                            = 16#0.100000#E-4
--               float'small  = 2.0 ** (-85) [LRM 3.5.8]
--                            = 2#1.000_0000_0000_0000_0000_0000#E-85
--                            = 16#0.800_000_0#E-21
--             float'large  = (2.0 ** 84) * (1.0 - 2.0 ** (-21)) [LRM 3.5.8]
--                            = 2#1.111_1111_1111_1111_1111_1#E+83
--                            = 16#0.FFF_FF8_0#E+21
--           float'safe_emax  = 127 [LRM 3.5.7, 3.5.8]
--          float'safe_small  = 2.0 ** (-126) [LRM 3.5.7]
--                            = 2#1.000_0000_0000_0000_0000_0000#E-126
--                            = 16#0.400_000#E-31
--      float'safe_large  = (2.0 ** 128) * (1.0 - 2.0 ** (-21)) [LRM 3.5.7]
--                            = 2#1.111_1111_1111_1111_1111_1#E+127
--                            = 16#0.FFF_FF8#E+32
--               float'first  = -float'last
--                float'last  = (2.0 ** 128) * (1.0 - 2.0 ** (-24))
--                            = 2#1.111_1111_1111_1111_1111_1111#E+127
--                            = 16#0.FFF_FFF#E+32
--                            ~ 3.40_282_347E+38
--      float'machine_radix  = 2
--   float'machine_mantissa  = 24
```

```
vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
(10)  File names
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
```

As SEQUENTIAL_IO and DIRECT_IO are not supported on the target(s), there are no file name conventions on the target configuration(s).

```
---------------------------
-- FAST INTERRUPT ENTRIES --
---------------------------


  --Prompt Interrupt Entry:

  PROMPT                        ,
    --This is a Fast Interrupt Entry, invoked by an interrupt other than
    --NMI or Single Step, whose accept body receives control after an
    --interrupt more quickly than an ordinary interrupt entry but more
    --slowly than a Quick or a Non-Maskable Interrupt Entry.  The accept
    --body may make conditional entry calls to entries that have been
    --declared to be Trivial Entries by means of the pragma
    --TRIVIAL_ENTRY.

    --When this kind of interrupt entry occurs, the state of the 8087
    --Numeric Data Processor will always be saved as part of the context
    --of the interrupted task, because the normal task-switching
    --mechanism will attempt to restore it before resuming the
    --interrupted task.


  --Note: In the following constant names, "NDP" stands for "Numeric Data
  --Processor," i.e., the Intel 8087.

  --Quick Interrupt Entries:

  SIMPLE_QUICK                  ,
    --This is a Quick Interrupt Entry, invoked by an interrupt other than
    --NMI or Single Step, whose accept body makes no entry calls.

  NO_NDP_SIMPLE_QUICK           ,
    --This is a Quick Interrupt Entry, invoked by an interrupt other than
    --NMI or Single Step, whose accept body makes no entry calls.

    --It differs from SIMPLE_QUICK only in that the state of the 8087
    --Numeric Data Processor is neither saved nor restored during
    --interrupt delivery.

  SIGNALLING_QUICK              ,
    --This is a Quick Interrupt Entry, invoked by an interrupt other than
    --NMI or Single Step, whose accept body may make conditional entry
    --calls to entries that have been declared to be Trivial Entries by
    --means of the pragma TRIVIAL_ENTRY.

    --When this kind of interrupt entry occurs, the state of the 8087
    --Numeric Data Processor will always be saved as part of the context
    --of the interrupted task, because the normal task-switching
    --mechanism will attempt to restore it before resuming the
    --interrupted task.


  --Non-Maskable Interrupt Entries:

  NON_MASKABLE                  ,
```

APPENDIX C

TEST PARAMETERS

Certain tests in the ACVC make use of implementation-dependent values, such as the maximum length of an input line and invalid file names. A test that makes use of such values is identified by the extension .TST in its file name. Actual values to be substituted are represented by names that begin with a dollar sign. A value must be substituted for each of these names before the test is run. The values used for this validation are given below.

Name and Meaning	Value
$BIG_ID1 Identifier the size of the maximum input line length with varying last character.	<1..119 => 'A', 120 => '1'>
$BIG_ID2 Identifier the size of the maximum input line length with varying last character.	<1..119 => 'A', 120 => '2'>
$BIG_ID3 Identifier the size of the maximum input line length with varying middle character.	<1..59 => 'A', 60 => '3', 61..120 => 'A'>
$BIG_ID4 Identifier the size of the maximum input line length with varying middle character.	<1..59 => 'A', 60 => '4', 61..120 => 'A'>
$BIG_INT_LIT An integer literal of value 298 with enough leading zeroes so that it is the size of the maximum line length.	<1..117 => '0', 118..120 => '298'>
$BIG_REAL_LIT A universal real literal of value 690.0 with enough leading zeroes to be the size of the maximum line length.	<1..114 => '0', 115..120 => '69.0E1'>

$BIG_STRING1 A string literal which when catenated with BIG_STRING2 yields the image of BIG_ID1.	<1..60 => 'A'>
$BIG_STRING2 A string literal which when catenated to the end of BIG_STRING1 yields the image of BIG_ID1.	<1..59 => 'A', 60 => 'A'>
$BLANKS A sequence of blanks twenty characters less than the size of the maximum line length.	<1..100 => ' '>
$COUNT_LAST A universal integer literal whose value is TEXT_IO.COUNT'LAST.	2_147_483_647
$FIELD_LAST A universal integer literal whose value is TEXT_IO.FIELD'LAST.	2_147_483_647
$FILE_NAME_WITH_BAD_CHARS An external file name that either contains invalid characters or is too long.	BAD-CHARS^#.%!X
$FILE_NAME_WITH_WILD_CARD_CHAR An external file name that either contains a wild card character or is too long.	WILD-CHAR*.NAM
$GREATER_THAN_DURATION A universal real literal that lies between DURATION'BASE'LAST and DURATION'LAST or any value in the range of DURATION.	75_000.0
$GREATER_THAN_DURATION_BASE_LAST A universal real literal that is greater than DURATION'BASE'LAST.	131_073.0
$ILLEGAL_EXTERNAL_FILE_NAME1 An external file name which contains invalid characters.	BADCHAR^@.~!

$ILLEGAL_EXTERNAL_FILE_NAME2 An external file name which is too long.	THIS-FILE-NAME-WOULD-BE-PERFECTLY -LEGAL-IF-IT-WERE-NOT-SO-LONG-- IT-HAS-NEARLY-ONE-HUNDRED-SIXTY- CHARACTERS
$INTEGER_FIRST - A universal integer literal whose value is INTEGER'FIRST.	-2_147_483_648
$INTEGER_LAST A universal integer literal whose value is INTEGER'LAST.	2_147_483_647
$INTEGER_LAST_PLUS_1 A universal integer literal whose value is INTEGER'LAST + 1.	2_147_483_648
$LESS_THAN_DURATION A universal real literal that lies between DURATION'BASE'FIRST and DURATION'FIRST or any value in the range of DURATION.	-75_000.0
$LESS_THAN_DURATION_BASE_FIRST A universal real literal that is less than DURATION'BASE'FIRST.	-131_073.0
$MAX_DIGITS Maximum digits supported for floating-point types.	15
$MAX_IN_LEN Maximum input line length permitted by the implementation.	120
$MAX_INT A universal integer literal whose value is SYSTEM.MAX_INT.	2_147_483_647
$MAX_INT_PLUS_1 A universal integer literal whose value is SYSTEM.MAX_INT+1.	2_147_483_648
$MAX_LEN_INT_BASED_LITERAL A universal integer based literal whose value is 2#11# with enough leading zeroes in the mantissa to be MAX_IN_LEN long.	<1..2 => '2:', 3..117 => '0', 118..120 => '11:'>

$MAX_LEN_REAL_BASED_LITERAL A universal real based literal whose value is 16:F.E: with enough leading zeroes in the mantissa to be MAX_IN_LEN long.	<1..3 => '16:', 4..116 => '0', 117..120 => 'F.E:'>
$MAX_STRING_LITERAL A string literal of size MAX_IN_LEN, including the quote characters.	<1 => '"', 2..119 => 'A', 120 => '"'>
$MIN_INT A universal integer literal whose value is SYSTEM.MIN_ INT.	-2_147_483_648
$NAME A name of a predefined numeric type other than FLOAT, INTEGER, SHORT_FLOAT, SHORT_INTEGER, LONG_FLOAT, or LONG_INTEGER.	No_Such_Type
$NEG_BASED_INT A based integer literal whose highest order nonzero bit falls in the sign bit position of the representation for SYSTEM.MAX_INT.	16#FFFFFFFE#

APPENDIX D

WITHDRAWN TESTS

Some tests are withdrawn from the ACVC because they do not conform to the Ada Standard. The following 28 tests had been withdrawn at the time of validation testing for the reasons indicated. A reference of the form "AI-ddddd" is to an Ada Commentary.

B28003A: A basic declaration (line 36) wrongly follows a later declaration.

E28005C: This test requires that 'PRAGMA LIST (ON);' not appear in a listing that has been suspended by a previous "pragma LIST (OFF);"; the Ada Standard is not clear on this point, and the matter will be reviewed by the ARG.

C34004A: The expression in line 168 wrongly yields a value outside of the range of the target type T, raising CONSTRAINT_ERROR.

C35502P: Equality operators in lines 62 & 69 should be inequality operators.

A35902C: Line 17's assignment of the nominal upper bound of a fixed-point type to an object of that type raises CONSTRAINT_ERROR, for that value lies outside of the actual range of the type.

C35904A: The elaboration of the fixed-point subtype on line 28 wrongly raises CONSTRAINT_ERROR, because its upper bound exceeds that of the type.

C35904B: The subtype declaration that is expected to raise CONSTRAINT_ERROR when its compatibility is checked against that of various types passed as actual generic parameters, may in fact raise NUMERIC_ERROR or CONSTRAINT_ERROR for reasons not anticipated by the test.

C35A03E, & R: These tests assume that attribute 'MANTISSA returns 0 when applied to a fixed-point type with a null range, but the Ada Standard doesn't support this assumption.

C37213H: The subtype declaration of SCONS in line 100 is wrongly expected to raise an exception when elaborated.

C37213J: The aggregate in line 451 wrongly raises CONSTRAINT_ERROR.

C37215C, E, G, H:	Various discriminant constraints are wrongly expected to be incompatible with type CONS.
C38102C:	The fixed-point conversion on line 23 wrongly raises CONSTRAINT_ERROR.
C41402A:	'STORAGE_SIZE is wrongly applied to an object of an access type.
C45332A:	The test expects that either an expression in line 52 will raise an exception or else MACHINE_OVERFLOWS is FALSE. However, an implementation may evaluate the expression correctly using a type with a wider range than the base type of the operands, and MACHINE_OVERFLOWS may still be TRUE.
C45614C:	REPORT.IDENT_INT has an argument of the wrong type (LONG_INTEGER).
E66001D:	Wrongly allows either the acceptance or rejection of a parameterless function with the same identifier as an enumeration literal; the function must be rejected (see Commentary AI-00330).
A74106C, C85018B, C87B04B, CC1311B:	A bound specified in a fixed-point subtype declaration lies outside of that calculated for the base type, raising CONSTRAINT_ERROR. Errors of this sort occur re lines 37 & 59, 142 & 143, 16 & 48, and 252 & 253 of the four tests, respectively (and possibly elsewhere).
BC3105A:	Lines 159..168 are wrongly expected to be illegal; they are legal.
AD1A01A:	The declaration of subtype INT3 raises CONSTRAINT_ERROR for implementations that select INT'SIZE to be 16 or greater.
CE2401H:	The record aggregates in lines 105 & 117 contain the wrong values.
CE3208A:	This test expects that an attempt to open the default output file (after it was closed) with mode IN_FILE raises NAME_ERROR or USE_ERROR; by Commentary AI-00048, MODE_ERROR should be raised.